

**UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS**

**FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA**

**E.A.P DE INGENIERIA DE SISTEMAS**

**Ingeniería inversa aplicado a sistemas desarrollados  
con programación orientada a objetos para obtener la  
documentación**

**TESIS**

**Para optar el Título Profesional de Ingeniero de Sistemas**

**AUTORES:**

**Jessica Jahany Acevedo Ricse**

**Elmer Emilio Puma Falcón**

**Lima-Perú**

**2007**

### **DEDICATORIA**

A nuestra familia por su apoyo constante e incondicional,  
a nuestros maestros y amigos que han contribuido con  
nuestro desarrollo personal y profesional.

## **INDICE**

<b>INTRODUCCION</b>	<b>9</b>
 <b>CAPITULO I</b>	
<b>1. PLANTEAMIENTO DEL PROBLEMA</b>	<b>12</b>
1.1. Fundamentación del problema	12
1.2. Descripción de la Realidad	14
1.2.1. La falta de documentación	14
1.2.2. La importancia del mantenimiento de software	15
1.3. Antecedentes del Problema	20
1.4. Justificación de la Investigación	24
1.5. Importancia de la Investigación	25
1.6. Limitaciones de la Investigación	25
 <b>CAPITULO II</b>	
<b>2. FORMULACION DEL PROBLEMA</b>	<b>26</b>
2.1. Objetivos	26
2.1.1. Objetivos Generales	26
2.1.2. Objetivos Específicos	26
2.2. Definición del Problema	26
 <b>CAPITULO III</b>	
<b>3. MARCO TEORICO CONCEPTUAL</b>	<b>29</b>
3.1. Antecedentes de la Investigación	29
3.2. Bases Teóricas	31

3.2.1. Mantenimiento de Software	31
3.2.1.1. Tipos de Mantenimiento de Software	33
3.2.1.2. La Gestión del mantenimiento de software	35
3.2.1.3. Soluciones Técnicas	38
3.2.2. Ingeniería Inversa	39
3.2.2.1. Objetivos y Beneficios	40
3.2.2.2. Elementos	42
3.2.2.3. Fases	44
3.2.2.4. Áreas	52
3.2.2.5. Inconvenientes	53
3.2.2.6. Herramientas CASE	53
3.3. Definiciones Básicas	54
3.3.1. ingeniería Inversa	54
3.3.2. Reingeniería	54
3.3.3. UML	54
3.3.4. Programación Orientada a Objetos	55

## **CAPITULO IV**

<b>4. METODOLOGIA PROPUESTA</b>	<b>56</b>
4.1. Estudio del sistema existente	58
4.2. Recuperación arquitectónica	64
4.3. Documentación de los casos de uso	83

## **CAPITULO V**

<b>5. ESTADO DEL ARTE</b>	<b>91</b>
---------------------------	-----------

5.1. Ingeniería Inversa en casos de uso UML	91
5.2. Ingeniería Inversa basado en diseño de patrones	94
 <b>CAPITULO VII</b>	
<b>6. CONCLUSIONES</b>	<b>98</b>
 <b>CAPITULO VII</b>	
<b>7. RECOMENDACIONES</b>	<b>99</b>
 <b>CAPITULO VIII</b>	
<b>8. REFERENCIAS BIBLIOGRAFICAS</b>	<b>100</b>
 <b>CAPITULO IX</b>	
<b>9. ANEXOS</b>	<b>103</b>
ANEXO N° 01	103
ANEXO N° 02	110

## INDICE DE FIGURAS

Figura 1. Distribución del costo del ciclo de vida	16
Figura 2. Comparación del costo desarrollo y mantenimiento	17
Figura 3. Coste relativo aproximado de detectar y corregir defectos	19
Figura 4. El modelo de vistas “4+1”	29
Figura 5. Fases de la Ingeniería Inversa	45
Figura 6. Del código fuente hacia el modelo conceptual	56
Figura 7. Pantalla de Ingreso al sistema TravelPlus	60
Figura 8. Pantalla de Listado de Agencias y Usuarios	60
Figura 9. Pantalla de Ingreso de una nueva agencia	61
Figura 10. Pantalla de modificación de datos de una agencia	61
Figura 11. Pantalla de Búsqueda de Hoteles	62
Figura 12. Pantalla de Hoteles Encontrados	62
Figura 13. Pantalla de Detalle de Hotel	63
Figura 14. Pantalla de Ingreso de datos de la reservación	63
Figura 15. Pantalla de Constancia de reservación	64
Figura 16. Descripción del modelo desde cinco vistas	65
Figura 17. Gestión del sistema TravelPlus	67
Figura 18. Diagrama de caso de uso – Gestión Agencia de Viajes	71
Figura 19. Diagrama de caso de uso – Gestión Usuarios	72
Figura 20. Diagrama de caso de uso – Reservación de Habitaciones	72
Figura 21. Diagrama de Clases de la capa de negocio	76
Figura 22. Clase ReservationDetail – Atributos y Métodos	78
Figura 23. Diagrama de Secuencia – Ingresar Nueva Agencia	80
Figura 24. Diagrama de Colaboración – Ingresar Nueva Agencia	81

Figura 25. Concepto del enrejado	93
Figura 26. Trabajando con paquetes	93
Figura 27. Visión general del ambiente SPOOL	95
Figura 28. Interfaz grafica del ambiente SPOOL	97

## **RESUMEN**

# **INGENIERIA INVERSA APLICADO A SISTEMAS DESARROLLADOS CON PROGRAMACION ORIENTADA A OBJETOS PARA OBTENER LA DOCUMENTACION**

Jessica Jahany Acevedo Ricse

Elmer Emilio Puma Falcon

**Septiembre – 2007**

**Asesor :** Carlos Ruiz de la Cruz Melo  
**Grado a obtener :** Ingeniero de Sistemas

---

El presente trabajo tiene por objetivo proponer una metodología basada en la ingeniería inversa con el fin de recuperar la documentación funcional y las especificaciones de diseño de sistemas desarrollados con metodología orientada a objetos, facilitando con esto el mantenimiento del mismo. Nosotros presentamos un enfoque basado en vistas de diseño utilizando el estándar UML, apoyándonos en una Herramienta CASE para la recuperación de diagramas (diagrama de clases) desde el código fuente.

### **Palabras claves:**

Sistemas  
Software  
Ingeniería Inversa  
UML  
Herramienta CASE



## **ABSTRACT**

# **REVERSE ENGINEERING APPLIED TO SYSTEM DEVELOPED WITH OBJECT ORIENTED PROGRAMMING FOR OBTAINING DOCUMENTATION**

Jessica Jahany Acevedo Ricse

Elmer Emilio Puma Falcon

**September – 2007**

<b>Adviser</b>	:	<b>Carlos Ruiz de la Cruz Melo</b>
<b>Title to obtain</b>	:	System Engineer

---

The present work has as objective proposes a methodology based on reverse engineering with the purpose of recovering the functional documentation and the design specifications of system developed with Object Oriented Methodology facilitating with this the maintenance of itself. We presented an approach based on design views using standard UML, supporting in a Tool CASE for the recovery of diagrams (diagram of classes) from the source code.

### **Keywords:**

Systems  
Software  
Reverse Engineering  
UML  
Tool CASE

## INTRODUCCION

Todo sistema de información se construye para solucionar problemas presentes. Un cambio en las condiciones del entorno organizacional puede producir un cambio en los objetivos de la organización y, en consecuencia, las reglas de la organización también se modifican. Muchos sistemas en funcionamiento, deben someterse a un mantenimiento apremiante y constante. Para eso es necesario contar con las herramientas necesarias para realizar un adecuado control de cambios y mantenimiento del sistema.

En la actualidad muchas empresas (especialmente las pequeñas y medianas empresas) que desarrollan software para otras empresas o para ellas mismas no tienen conciencia de lo importante que es el mantenimiento y tratan de “ahorrar” costos al no invertir en tareas de documentación de sus sistemas, presentando así una inadecuada y desactualizada documentación, y en algunos casos una documentación nula, que desemboca con el tiempo en una degradación de la aplicación y, por ende, en un servicio deficiente para el cliente y/o usuario. Existen también muchos sistemas de software que son complejos o antiguos y que su información no es fácilmente disponible, y sin el suficiente entendimiento del sistema este mantenimiento no puede ser realizado.

Múltiples estudios señalan que el mantenimiento es la parte más costosa del ciclo de vida del software. Estadísticamente está comprobado que el coste

de mantenimiento de un producto software a lo largo de toda su vida útil supone mas del doble que los costes de su desarrollo.

El propósito del presente trabajo es proponer una metodología de ingeniería inversa como una solución para entender el sistema existente, a través de la recuperación del diseño y la especificación de sus requisitos, contribuyendo así con el entendimiento de la funcionalidad del sistema que se va hacer mantenimiento. El trabajo esta dividido en los siguientes capítulos:

El Capitulo I presenta el planteamiento del problema desde sus antecedentes, describe la realidad actual, justificación, importancia y limitaciones de la investigación..

El Capitulo II define los Objetivos Generales y Específicos que pretende alcanzar la presente investigación y se define el problema.

El Capitulo III detalla los antecedentes de la investigación y las bases teóricas la cual se subdivide en dos secciones: Mantenimiento del software e Ingeniería Inversa.

El Capitulo IV define la metodología propuesta, la cual es aplicada en un caso practico.

El Capitulo V desarrolla el estado del arte en dos temas relacionados con el trabajo de tesina, los temas son: Ingeniería Inversa en casos de uso UML e Ingeniería Inversa basada en diseño de patrones.

El capitulo VI recoge las conclusiones de la investigación realizada y finalmente en el capitulo VII se entregan recomendaciones, las cuales se han construido en base a la investigación realizada.

## **CAPITULO I**

### **1. PLANTEAMIENTO DEL PROBLEMA**

El mantenimiento de software es una de las actividades más comunes en la Ingeniería de Software y es el proceso de mejora y optimización del software desplegado (es decir; revisión del programa), así como también corrección de los defectos.

El mantenimiento de software existente puede dar cuenta de más del 60 por 100 de las inversiones efectuadas por una organización de desarrollo, y ese porcentaje sigue ascendiendo a medida que se produce más software.

#### **1.1. Fundamentación del problema**

Es casi seguro decir que el mantenimiento del software consume el 60% a 80% del coste de un sistema y probablemente sea la parte más importante del ciclo de vida del software (a parte de ser la más olvidada). No debemos pensar que el mantenimiento solo consiste en reparar todo aquello que deja de funcionar por un defecto de fabricación o por el desgaste asociado al paso del tiempo (la naturaleza del software hace que nunca se rompa por desgaste), sino que también incluyen el mejoramiento y la adición de nueva funcionalidad.

De todo los gastos asociados al mantenimiento del software (que viene a suponer un 60% de su coste total), un 80% corresponde a la realización de

mejoras y nuevas funcionalidades, mientras que un 20% esta asociado a la corrección de defectos.

Ahora, no es difícil adivinar que sin el conocimiento del sistema existente y sin una documentación adecuada de este, estos costos se incrementarían considerablemente impactando en forma abrupta en la calidad y atención al cliente y/o usuario.

Citando como ejemplo al Perú, existen muchas empresas que no cuentan con un marco teórico común que puedan ser usados por todas las personas que participan en el desarrollo de los proyectos, muchas de estas empresas también priorizan la fase de construcción (programación) y despliegue con la finalidad de cumplir compromisos con el cliente, posponiendo y olvidando así el registro y documentación de los nuevos requerimientos.

A pesar de la experiencia de los analistas programadores, la planificación de los costes del mantenimiento se presenta como una situación de incertidumbre, en la que no se sabe que va a pasar, ya que sería como predecir el futuro. Esta situación es similar a un iceberg, del cual tan sólo podemos percibir una pequeña parte del problema, escondiéndose una gran cantidad de problemas y costes adicionales e inesperados bajo la superficie.

Como vemos el mantenimiento del software es una etapa crítica, sin una adecuada documentación y sin la conciencia de la importancia de esto, la

mantenibilidad de un sistema podría determinar el éxito o fracaso del sistema, y porque no decirlo también de la empresa.

## **1.2. Descripción de la Realidad**

En la actualidad existen muchas empresas con una serie de problemas que hacen difícil la tarea de mantener sus sistemas, entre ellas podemos mencionar: la falta de documentación, y la falta de un marco teórico y técnico común que permita a los participantes del proyecto construir aplicaciones legibles, flexibles y mantenibles.

### **1.2.1. La falta de documentación**

El desarrollo de aplicaciones se ha convertido en una tarea compleja que involucra un gran número de recursos (tanto humanos como materiales); resulta pues de vital importancia la adopción de métodos a fin de guiar la construcción, mantenimiento y evolución de la aplicación a través de su ciclo de vida.

La fase de mayor duración y mayor costo de un programa o aplicación, y durante la que se deben hacer toda clase de cambios, alteraciones y mejoras, es la del mantenimiento.

Es aquí donde aparece el dilema clásico: Los desarrolladores que crearon el código original ya no están en la empresa, y un nuevo programador debe hacer frente a "la bestia" (un listado de código ilegible, sin comentario alguno y con altas probabilidades de convertirse

en la peor pesadilla del recién contratado, o de los usuarios si éste se va por el camino de los parches).

El problema que se nos presenta es que en múltiples ocasiones, el nuevo mantenedor se encuentra con un producto de software con poca o ninguna documentación, el cual trae como consecuencias:

Para el mantenedor:

- Dificultad para seguir la evolución del software a través de varias versiones al no existir documentación sobre los cambios
- Dificultad para seguir el proceso por el que se construyó el software
- Dificultad para comprender un programa ajeno
- Dificultad para contactar con los desarrolladores

Para la empresa:

- Costo del mantenedor
- Tiempo requerido por el mantenedor para comprender el sistema

### **1.2.2. La importancia del mantenimiento de software**

Aun cuando son las últimas en el ciclo de vida del software<sup>1</sup>, las actividades de mantenimiento no son las menos importantes, muy al contrario, el mantenimiento del software se ha convertido en la principal actividad debido a su repercusión económica, temporal y de recursos.

---

<sup>1</sup> Un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto software.



En la figura 1 podemos ver como esta distribuido el costo durante el ciclo de vida del sistema.

### **Distribución del Costo del Ciclo de Vida (Rock-Evans y Hales, 1990)**

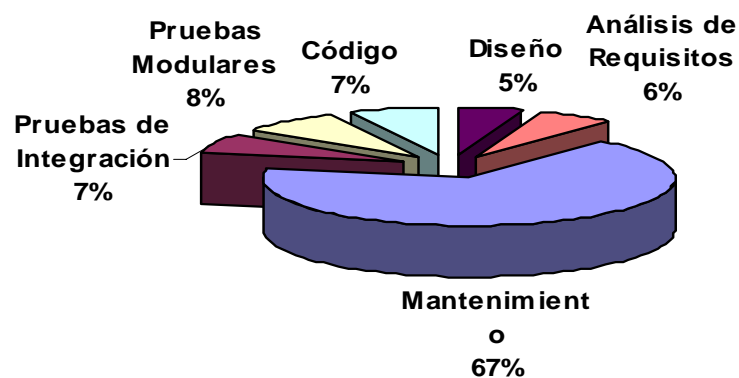


Figura 1. Distribución del costo del ciclo de vida

Múltiples estudios señalan que el mantenimiento es la parte más costosa del ciclo de vida del software. Estadísticamente está comprobado que el coste de mantenimiento de un producto software a lo largo de toda su vida útil supone mas del doble que los costes de su desarrollo. La tendencia es creciente con el paso del tiempo (ver Figura 2):

Referencia	Fechas	Mantenimiento %
[Pressman, 1993]	Años 70	35 – 40
[Lientz y Swanson, 1980]	1976	60
[Pigoski, 1997]	1980 - 1984	55
[Presuman, 1993]	Años 80	60
[Rock-Evans y Hales, 1990]	1987	67
[Schach, 1990]	1987	67
[Pigoski, 1997]	1985 – 1989	75
[Frazer, 1992]	1990	80
[Pressman, 1993]	Años 90	90

Tabla 1. Evolución de los costes de mantenimiento

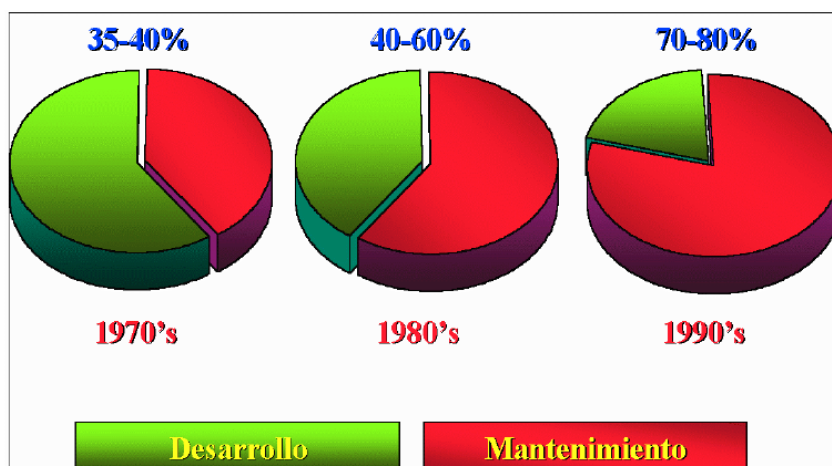


Figura 2. Comparación del costo desarrollo y mantenimiento

***El Mantenimiento de Software supone, cada vez más, un mercado importantísimo.***

¿Cuales son las causas del alto coste de mantenimiento de software?

Son varias las causas de que en la mayoría de las organizaciones actuales se requiera mucho trabajo de mantenimiento [6]:

- Una gran cantidad del software que existe actualmente ha sido desarrollado hace más de 10 años. Aunque estos programas fuesen creados utilizando las mejores técnicas de diseño y codificación existentes en su momento (la mayoría no lo fueron), se construyeron con restricciones de tamaño y espacio de almacenamiento y se desarrollaron con herramientas tecnológicamente desfasadas.
- Estos programas han sufrido una o varias migraciones a nuevas plataformas o sistemas operativos.
- Y han experimentado múltiples modificaciones para mejorarlos y adaptarlos a las nuevas necesidades de los usuarios.
- Todos estos cambios se realizaron sin tener en cuenta la arquitectura general del sistema (no se aplicaron técnicas de ingeniería inversa<sup>2</sup> o reingeniería<sup>3</sup>).

El resultado de todo lo anterior, es la existencia de sistemas software con una baja calidad:

- diseño pobre de las estructuras de datos
- mala codificación
- lógica defectuosa
- documentación escasa.

---

<sup>2</sup> La ingeniería inversa realiza un análisis de un sistema de software para conseguir especificar su documentación.

<sup>3</sup> consiste en el examen y modificación de un sistema para reconstruirlo de una nueva forma.

Pero que tienen que seguir funcionando, y por tanto, tienen que ser mantenidos:

Baja calidad => mayores costes de mantenimiento.

Otra causa directa de los grandes costes del Mantenimiento de Software es que el coste relativo de reparar un defecto aumenta considerablemente en las últimas etapas del ciclo de vida del software, de forma que la relación entre el coste de detectar y reparar un defecto en la fase de análisis de requisitos y en la fase de mantenimiento es de 1 a 100 respectivamente (ver Figura 3).

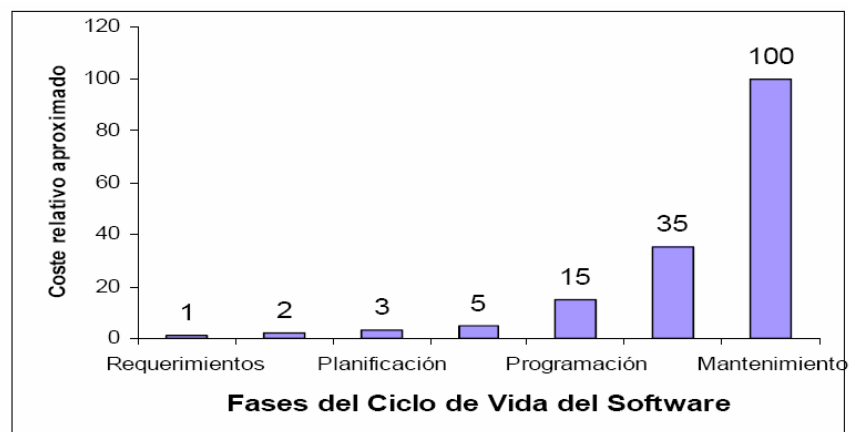


Figura 3. Coste relativo aproximado de detectar y corregir defectos

Algunas de las razones por las que es menos costoso detectar y corregir un error durante las etapas iniciales del ciclo de vida que durante las etapas últimas son:

- Es más fácil cambiar la documentación (por ejemplo, los documentos de especificación o de diseño) que modificar el código.
- Un cambio durante una fase tardía puede requerir que sea modificada la documentación de todas las fases anteriores.
- Es más fácil encontrar un defecto durante la fase en la cual se ha introducido el defecto que tratar de detectar y corregir los efectos provocados por el defecto en una fase posterior.
- La causa de un defecto puede esconderse en la inexistencia o falta de actualización de los documentos de especificación o diseño.

### **1.3. Antecedentes del Problema**

Ha habido una serie de antecedentes que han llevado a las empresas a tomar decisiones costosas y que no han resuelto sus problemas y muy por el contrario han sido grandes errores estratégico. Entre estos antecedentes encontramos:

#### **1.3.1. Sistemas Heredados**

Las compañías gastan mucho dinero en sistemas de software y, para obtener un beneficio de esa inversión, el software debe utilizarse por varios años. El tiempo de vida de los sistemas de software es muy variable, pero muchos sistemas grandes se utilizan hasta por más de 20 años. Muchos de estos sistemas antiguos aún son importantes para los negocios. Esto es, los negocios cuentan con los servicios suministrados

por el software y cualquier falla en estos servicios tendría un efecto serio en el funcionamiento de los negocios. Estos sistemas antiguos reciben el nombre de sistemas heredados. Por supuesto, estos sistemas heredados no son los sistemas que se entregaron originalmente.

Los factores externos e internos, como el estado de las economías nacional e internacional, los mercados cambiantes, los cambios en las leyes, los cambios de administración y la reorganización estructural, conducen a que los negocios experimenten cambios continuos. Estos cambios generan o modifican los requerimientos del software por lo que todos los sistemas de software inevitablemente cambian conforme cambian los negocios.

Por lo tanto, los sistemas heredados incorporan un gran número de cambios hechos a lo largo de varios años. Muchas personas diferentes estuvieron involucradas al realizar estos cambios y es inusual para cualquier persona tener un conocimiento completo del sistema. Los negocios por lo regular reemplazan su equipo y maquinaria con sistemas más modernos. Sin embargo, desechar los sistemas heredados y reemplazados con software moderno conduce a riesgos de negocios significantes. Reemplazar un sistema heredado es una estrategia de negocios riesgosa por varias razones:

- Rara vez existe una especificación completa de los sistemas heredados. La especificación original se perdió. Si existe una especificación, no es probable que tenga los detalles de todos los cambios hechos en el sistema. Por lo tanto, no existe alguna

forma directa de especificar un nuevo sistema que sea funcionalmente idéntico al sistema que se utiliza.

- Los procesos de negocios y las formas en que los sistemas heredados operan a menudo están intrincadamente entrelazados. Estos procesos se diseñaron para aprovechar los servicios del software y evitar sus debilidades. Si el sistema se reemplaza, estos procesos también tendrán que cambiar, con costos y consecuencias impredecibles.
- Las reglas de negocios importantes están contenidas en el software y no están mentadas en algún otro lugar. Una regla de negocios es una restricción que aparece en algunas funciones de negocios y romper esa restricción puede tener consecuencias impredecibles para los negocios. Por ejemplo, las reglas para valorar el riesgo de la aplicación de una política de una compañía de seguros pueden estar contempladas en su software.
- El desarrollo de nuevo software es por sí mismo es riesgoso ya que existen problemas mas inesperados con un nuevo sistema. Puede ser que no se entregue a tiempo con el precio esperado.

### **1.3.2. Reescribir código**

Los programadores son, en lo más profundo, arquitectos, y la primera cosa que quieren hacer cuando llegan a un sitio es nivelar el terreno a fuerza de tractores y construir algo magnífico. No nos gusta la renovación incremental: hacer pequeños ajustes, mejoras.

Hay siempre una sutil razón por la que los programadores siempre quieren desechar el código y empezar de nuevo. La razón es que piensan que el viejo código es una ruina. Y aquí está la observación interesante: *probablemente están equivocados*. La razón por la que piensan que el viejo código es una ruina es por una ley capital y fundamental de la programación: “*Es más difícil leer código que escribirlo*”. Esta es la razón por la que todo el mundo en un equipo de desarrollo tiene una función diferente que le gusta usar para separar cadenas en arreglos de cadenas por citar un ejemplo. Escriben su propia función porque es más fácil y más divertido que adivinar cómo funciona la antigua función.

La idea de que el nuevo código es mejor que el viejo es a todas luces absurdo. El viejo código ha sido *usado*. Ha sido *probado*. Muchos errores han sido encontrados, y han sido *arreglados*. No hay nada malo en ello. No adquiere errores en el disco duro simplemente por estar ahí. Al contrario. ¿Se supone que el software es como un viejo carro, que se oxida simplemente estando en el garaje? ¿Es el software como un osito de peluche que es de mala calidad si no está hecho *todo de material nuevo*?

Cuando desechas código y empiezas desde cero, estás desechando todo ese conocimiento. Todos aquellos arreglos de errores. Años de trabajo de programación.



## **1.4. Justificación de la Investigación**

El presente trabajo surgió como una necesidad percibida en diversas empresas que desarrollan software, y que no poseen ninguna documentación del mismo.

### **1.4.1. Justificación de carácter práctico**

Desde el punto de vista práctico, el estudio sugiere la aplicación de una moderna herramienta CASE<sup>4</sup> que soporte ingeniería inversa que contribuya a la obtención de los diseños a partir del código ya existente.

### **1.4.2. Justificación de carácter metodológico**

La investigación es importante metodológicamente porque se propone un estudio de las diferentes metodologías relacionadas a la ingeniería inversa, con el objetivo de crear una alternativa, plasmada en un modelo que se sugiere que sigan las empresas que venían desarrollando sus sistemas de software con programación orientada a objetos sin tomar en consideración el ciclo de vida del mismo.

## **1.5. Importancia de la Investigación**

Como se ha mencionado anteriormente el mantenimiento del software es la etapa más prolongada y la que más costos genera, no es difícil imaginar

---

<sup>4</sup> son aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software.

que estos costos serían mucho más grandes si no contamos con una documentación adecuada del sistema.

Con este estudio, se pretende brindar una metodología para obtener los requerimientos funcionales y las especificaciones de diseño de un sistema ya construido con Programación Orientada Objetos.

### **1.6. Limitaciones de la Investigación**

Teniendo este estudio como fuentes a las principales metodologías de la ingeniería de Software y a los estándares para el análisis, diseño y programación tales como UML y Programación Orientada a Objetos esta investigación queda limitada al estudio de solo sistemas desarrollados con Programación orientada objetos.

Para el caso de aplicación se construyó un sistema de reservación de habitaciones con fines prácticos (solo módulo de reserva y mantenimientos).

## **CAPITULO II**

### **2. FORMULACION DEL PROBLEMA**

#### **2.1. Objetivos**

Los objetivos que se persiguen al realizar este trabajo están clasificados en:

##### **2.1.1. Objetivos Generales**

El objetivo principal de este trabajo es proponer una metodología para generar la documentación funcional y el diseño o especificación de sistemas de software a partir de su código fuente a través de la ingeniería inversa.

##### **2.1.2. Objetivos Específicos**

- Proponer una metodología de ingeniería inversa.
- Identificar la herramienta CASE que se acomode a nuestra metodología.
- Analizar el código fuente del sistema a ejemplificar.

#### **2.2. Definición del Problema**

Desarrollar un software significa construirlo simplemente mediante su descripción. En un nivel más general, la relación existente entre un software y su entorno es clara ya que el software es introducido en el mundo de modo de provocar ciertos efectos en el mismo.

Aquellas partes del mundo que afectarán al software y que serán afectadas por él será el Dominio de Aplicación. Es allí donde los usuarios y/o clientes observarán si el desarrollo del software ha cumplido su propósito.

Una de las mayores deficiencias en la práctica de desarrollo de software es la poca atención que se presta al ciclo de vida del mismo (análisis, diseño, codificación, prueba y mantenimiento). En general los desarrolladores se centran en la solución del problema, dejando de lado las etapas previas y necesarias para su desarrollo, logran resolverla, la ponen en funcionamiento y continúan con el resto de problemas.

Dentro del proceso de desarrollo de un sistema es necesario la existencia de una documentación que sustente el desarrollo del mismo, el cual permitirá reducir costo y tiempo de su mantenimiento; se torna necesario porque no se puede garantizar la permanencia del desarrollador por siempre, cuando este ya no este y sin documentación alguna el nuevo desarrollador tendrá que enfrentarse a una serie de dificultades:

- Dificultad para seguir la evolución del software a través de varias versiones al no existir documentación sobre los cambios
- Dificultad para seguir el proceso por el que se construyó el software
- Dificultad para comprender un programa ajeno
- Dificultad para contactar con los desarrolladores

Es por eso que el resultado de esta tesina fue la propuesta de una metodología apoyada en la tecnología de ingeniería inversa que nos permita la recuperación de la documentación inexistente. Esta tecnología se basa en la recuperación diseño y la especificación, apoyándose también en una herramienta CASE.

Aunque han sido muy pocas las investigaciones sobre temas de ingeniería inversa de software, nosotros deseamos aportar y contribuir con su estudio.

Los análisis realizados en esta investigación y la metodología propuesta pueden contribuir con aquellas empresas y/o personas que deseen recuperar documentación de un sistema desarrollado, también es una ayuda útil para el proceso de reingeniería de software.

## CAPITULO III

### 3. MARCO TEORICO CONCEPTUAL

#### 3.1. Antecedentes de la Investigación

En un Congreso Regional de Ciencia y Tecnología NOA 2003, llevada a cabo en la Universidad Nacional de Catamarca, **Diana Palliotto y Gabriel Romano**, de la Universidad Nacional de Santiago del Estero - Facultad de Ciencias Exactas y Tecnologías, presentan el trabajo "La tecnología de la Ingeniería Inversa: Un método con UML, guiado por casos de uso y basado en el modelo de vistas 4+1". Ver Figura 4

El objetivo de ese trabajo fue presentar el modelo de vistas 4+1 como una alternativa para recuperar la arquitectura del software.

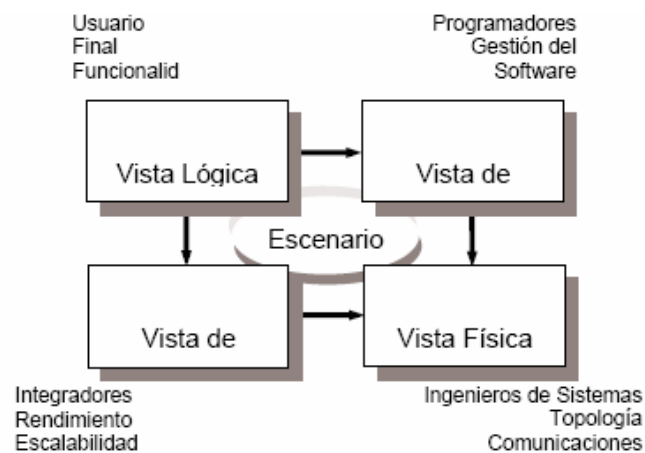


Figura 4. El modelo de vistas "4+1"

Otra investigación bibliografica, de la Universidad de las Americas Puebla, **Antonio Felipe Razo Rodríguez** [8], elaboró la tesis "Reingeniería para la implementación de un Web Feature Service".

El objetivo de ese trabajo fue utilizar la ingeniería inversa como parte de la reingeniería, propone su propio método para la recuperación de la documentación, apoyándose en casos de uso y diagrama de clases, las cuales se generaran a partir de herramientas CASE.

### **3.2. Bases Teóricas**

Las bases teóricas de este trabajo la podemos agrupar en:

#### **3.2.1. Mantenimiento de Software**

Mantenimiento es el conjunto de modificaciones de un producto software, realizado después de su desarrollo e implementación, para la corrección de errores, para mejorar el rendimiento u otros atributos del software, o para adaptar el producto a los cambios del entorno<sup>5</sup>. En el anexo N° 02 presentamos un grafico ilustrativo sobre este tema.

Así pues, el mantenimiento tiene lugar una vez que se distribuye el sistema para su explotación y es mucho más que una corrección de errores. A la vista de la definición aportada, podríamos describirlo mediante las siguientes cuatro actividades:

1. La corrección de errores, puesto que no es asumible que las pruebas en el desarrollo del software hayan descubierto todos ellos, y con el uso del sistema se producirán todavía algunos.
2. La segunda actividad que contribuye a la definición del mantenimiento se produce por el rápido cambio inherente a cualquier aspecto de la Informática. El hardware evoluciona rápidamente ofreciendo mejores entornos de producción y haciendo que el

---

<sup>5</sup> Según ANSI/IEEE 1219-1993



software desarrollado no soporte esta evolución y haya de modificarse.

3. La tercera actividad se produce cuando un software tiene éxito, puesto que a medida que se usa se reciben nuevas peticiones de los usuarios sobre la inclusión de nuevas funcionalidades o mejora de las existentes.
4. Por último, otra actividad de mantenimiento se da cuando se cambia el software para mejorar una futura facilidad de mantenimiento o fiabilidad.

Basándose en estas actividades se clasifican los diferentes tipos de mantenimiento.

A partir de la definición del mantenimiento del software, se pueden clasificar las causas que lo originan en dos tipos:

1. **Externas:** son las derivadas de necesidades no contempladas en el origen del sistema tales como: nuevas leyes o normativas, nuevas funcionalidades, etc.
2. **Internas:** son las derivadas por errores en el desarrollo o por la necesidad de actualizar un sistema que se va quedando obsoleto. Estas causas se deben en muchos casos, a que los sistemas no fueron desarrollados siguiendo una metodología de desarrollo.

### 3.2.1.1. Tipos de mantenimiento de software

Lientz y Swanson [1] proponen dividir en cuatro los tipos de mantenimiento:

1. **Adaptativo:** Para que el software se acople a un cambio de entorno, su objetivo es ajustar el sistema a cambios del entorno de trabajo.

Los cambios pueden afectar a:

- El sistema operativo, la arquitectura física del sistema informático.
- El entorno de desarrollo del software: Datos (sistema de archivos, bases de datos relacionales).
- Procesos (migrando a una nueva plataforma de desarrollo con componentes distribuidos, Java, ActiveX, etc.).

2. **Evolutivo:** Para suplir nuevos requerimientos. Su objetivo es mejorar o añadir nuevas funcionalidades requeridas por el usuario.

Se presenta dos tipos:

- Mantenimiento de Ampliación: orientado a la incorporación de nuevas funcionalidades.
- Mantenimiento de Eficiencia: que busca la mejora de la eficiencia de ejecución.

**3. Correctivo:** Para solucionar errores puntuales de programas.

Su objetivo es localizar y eliminar los posibles defectos de los programas.

Se presenta dos clases:

- Defecto: una característica del sistema con el potencial de causar un fallo.
- Fallo: ocurre cuando el comportamiento de un sistema es diferente del establecido en la especificación.

Tipos:

- Procesamiento.
- Rendimiento.
- Programación.
- Documentación.

**4. Preventivo:** Para prever la aparición de problemas. Su objetivo es mejorar las propiedades del software sin alterar sus especificaciones funcionales.

Modos:

- Incluir sentencias que comprueben la validez de los datos de entrada,
- Reestructurar los programas para mejorar su legibilidad, o

- Incluir nuevos comentarios que faciliten la posterior comprensión del programa

Se incluyen las tareas de modificación del software para que sea más fácilmente reutilizable (Generar bibliotecas de componentes).

### **3.2.1.2. La Gestión del Mantenimiento de Software**

Todo sistema está sujeto a cambios, tanto durante su desarrollo como cuando ya está en explotación. La gestión de estos cambios y su impacto en la configuración inicial del sistema es lo que abarca la “Gestión de la Configuración”, íntimamente relacionada con ella y parte fundamental de la misma es la Gestión del Mantenimiento del Software, que tratamos seguidamente.

Las tareas de mantenimiento del software comienzan mucho antes de que se haga una petición de mantenimiento. Inicialmente se debe establecer una organización de mantenimiento, se deben definir procedimientos de evaluación y de información, y se debe especificar una secuencia estandarizada de sucesos para cada petición de mantenimiento. Además, se debe establecer un sistema de registro de información de las actividades de mantenimiento y definir criterios de revisión y de evaluación. En definitiva, se debe organizar el mantenimiento.

Las peticiones de mantenimiento se dirigen hacia un controlador de mantenimiento, que remite cada petición a un supervisor del sistema para que la evalúe. Una vez hecha la evaluación, una autoridad de control de cambios (consejo de control de cambios) debe determinar las acciones a llevar a cabo.

Todas las peticiones de mantenimiento deben ser presentadas de forma estandarizada mediante un formulario de petición de mantenimiento, que es un documento generado externamente y que se usa como base para la planificación de las tareas de mantenimiento. Internamente, la organización de software desarrollará un informe de cambios en el software indicando: el esfuerzo requerido para satisfacer el formulario de petición de mantenimiento, la naturaleza de las modificaciones solicitadas y la prioridad de la petición.

La información relativa al desarrollo de las actividades de mantenimiento debe ser convenientemente registrada para poder ser evaluada y obtener así determinadas medidas del rendimiento del mantenimiento. Todo esto es una parte importante dentro de la Gestión del Mantenimiento y más en general de la Gestión de la Configuración, como se verá en el tema siguiente.

Finalmente, es importante tener en cuenta los efectos secundarios del mantenimiento, puesto que cada vez que se modifica el software,

la posibilidad de error aumenta. A estos efectos, se definen tres categorías de efectos secundarios:

1. **Efectos secundarios sobre el código**, que van desde los pequeños errores que se detectan y remedian durante las pruebas de regresión, hasta los problemas que pueden hacer que falle la operación del software.
2. **Efectos secundarios sobre los datos**, ya que durante el mantenimiento es corriente que se hagan cambios sobre determinados elementos de una estructura de datos o sobre la propia estructura en sí, y, en estos casos, puede ocurrir que el diseño del software no cuadre con los datos y aparezcan errores.
3. **Efectos secundarios sobre la documentación**, que se dan cuando no se reflejan los cambios del código fuente en la documentación del diseño y en los manuales de usuario. Estos efectos se pueden reducir revisando la configuración entera del software antes de lanzar una nueva versión.

Por último, otro aspecto que afecta al mantenimiento y su gestión es el mantenimiento de código ajeno, llamado así por referirse a programas que han sido desarrollados hace muchos años (diez o más) y en los que los miembros del equipo técnico actual no trabajaron en su desarrollo. Normalmente, para estos programas no se aplicó una metodología de desarrollo y, por tanto, existe un pobre

diseño arquitectónico y de datos, la documentación es incompleta y no suele existir un registro de los cambios anteriores

### **3.2.1.3. Soluciones Técnicas**

Son de dos tipos: herramientas y métodos.

Las herramientas sirven para soportar de forma efectiva a los métodos; han sido diseñadas para que el equipo de mantenimiento comprenda el programa y pruebe sus modificaciones asegurando que no han introducido errores. Estas herramientas son: formateador, analizador estático, estructurador, documentador, depurador interactivo, generador de datos de prueba y comparador.

Los principales métodos utilizados en el mantenimiento son:

**1. Reingeniería:** consiste en el examen y modificación de un sistema para reconstruirlo de una nueva forma. Rehacer algo que otro ha realizado tratando de reutilizar.

**2. Ingeniería Inversa:** proceso de analizar un sistema para identificar sus componentes e interrelaciones, así como crear representaciones del sistema en un nivel de abstracción más elevado. Reinterpretar un programa para documentarlo.

**3. Reestructuración del software:** consiste en la modificación del software para hacerlo más inteligible y más fácil de cambiar. No cambia el nivel de abstracción.

**4. Transformación de Programas:** método formal que parte de un programa ya existente para obtener un programa equivalente por medio de transformaciones sucesivas.

### **3.2.2. Ingeniería Inversa**

El término “Ingeniería Inversa” tiene sus orígenes en el mundo del hardware. Una empresa desensambla un producto de la competencia para intentar comprender los secretos del diseño y de la fabricación. Pues bien, la Ingeniería Inversa del software es bastante similar; si bien, en la mayoría de los casos el producto objeto de la misma no es de una empresa de la competencia sino, más bien, se trata de un trabajo propio, generalmente realizado muchos años atrás, y del que no existen especificaciones o éstas son muy incompletas.

La Ingeniería Inversa se ocupa de estudiar un sistema de información en el orden inverso establecido en el ciclo de vida habitual; esto es, partiendo del código fuente, se trata de identificar los componentes del sistema y las relaciones existentes entre ellos. Hasta su llegada, el ciclo de vida del software era, en teoría, en una sola dirección; ahora, se puede hablar de dos direcciones: forward o hacia adelante, que es la tradicional, y reverse o hacia atrás, que es la de la Ingeniería Inversa.

Para CHIKOKFSKY [9], Ingeniería Inversa es el proceso de analizar un sistema para identificar sus componentes e interrelaciones entre los mismos, y crear representaciones del sistema a un alto nivel de abstracción.



### **3.2.2.1. Objetivos y Beneficios de la Ingeniería Inversa**

El Objetivo primordial de la Ingeniería Inversa es proporcionar una base para el mantenimiento y futuros desarrollos. Este objetivo general se puede traducir en los siguientes objetivos parciales:

- Facilitar la reutilización.
- Proporcionar documentación que no existe, o actualizar la existente.
- Migrar a otra plataforma hardware o software, cuando sea necesario.
- Llevar el sistema bajo el control de un entorno CASE.

A la vista de estos objetivos, los sistemas candidatos a aplicarles la Ingeniería Inversa reúnen algunas de las siguientes características:

- Las especificaciones de diseño y la documentación, no existen o están incompletas.
- El código no es estructurado.
- El sistema necesita un excesivo mantenimiento correctivo.
- Algunos módulos se han hecho excesivamente complejos debido a los sucesivos cambios realizados en ellos.
- Se necesita una migración hacia una nueva plataforma de hardware o de software.

Aplicar la Ingeniería Inversa supone un enorme esfuerzo y, por tanto, se hace necesario evaluar exhaustivamente y siendo muy realistas,

los casos en los que es rentable su aplicación. A estos efectos, algunos aspectos que se deben considerar son los siguientes:

- Los programas que no se usan con mucha frecuencia no es probable que vayan a cambiar.
- Las herramientas de Ingeniería Inversa, generalmente herramientas CASE relativamente sofisticadas, sólo pueden utilizarse para una clase limitada de aplicaciones.
- El coste en esfuerzo y en dinero puede ser prohibitivo.

No obstante, la Ingeniería Inversa produce una serie de BENEFICIOS, que también deben ser considerados. Los más importantes son los siguientes:

- Disminuye los costes del mantenimiento, al reducir el tiempo dedicado a entender el software existente para, a su vez, entender el cambio que se desea realizar y para facilitar la solución del problema.
- Mejora la calidad del software, ya que los sistemas candidatos a aplicarles Ingeniería Inversa son, en general, sistemas de baja calidad.

- Aporta una mayor ventaja competitiva, dado que al facilitar el mantenimiento, permite un mayor poder de reacción ante los cambios solicitados por los usuarios.

### **3.2.2.2. Elementos de la Ingeniería Inversa**

Entre los elementos de la Ingeniería Inversa tenemos:

#### **1. El nivel de abstracción**

El nivel de abstracción de un proceso de ingeniería inversa y las herramientas que se utilizan para realizarlo aluden a la sofisticación de la información de diseño que se puede extraer del código fuente. El nivel de abstracción ideal deberá ser lo más alto posible. Esto es, el proceso de ingeniería inversa deberá ser capaz de derivar sus representaciones de diseño de procedimientos (con un bajo nivel de abstracción); y la información de las estructuras de datos y de programas (un nivel de abstracción ligeramente más elevado); modelos de flujo de datos y de control (un nivel de abstracción relativamente alto); y modelos de entidades y de relaciones (un elevado nivel de abstracción). A medida que crece el nivel de abstracción se proporciona al ingeniero del software información que le permitirá comprender más fácilmente estos programas.

## **2. La completitud**

La completitud de un proceso de ingeniería inversa alude al nivel de detalle que se proporciona en un determinado nivel de abstracción. En la mayoría de los casos, la completitud decrece a medida que aumenta el nivel de abstracción. Por ejemplo, dado un listado del código fuente, es relativamente sencillo desarrollar una representación de diseño de procedimientos completa. También se pueden derivar representaciones sencillas del flujo de datos, pero es mucho más difícil desarrollar un conjunto completo de diagramas de flujo de datos o un diagrama de transición de estados.

La completitud mejora en proporción directa a la cantidad de análisis efectuado por la persona que está efectuando la ingeniería inversa.

## **3. La Interactividad**

La interactividad alude al grado con el cual el ser humano se “integra” con las herramientas automatizadas para crear un proceso de ingeniería inversa efectivo. En la mayoría de los casos, a medida que crece el nivel de abstracción, la interactividad deberá incrementarse, o sino la completitud se verá reducida.

#### **4. La direccionalidad**

Si la direccionalidad del proceso de ingeniería inversa es monodireccional, toda la información extraída del código fuente se proporcionará a la ingeniería del software que podrá entonces utilizarla durante la actividad de mantenimiento. Si la direccionalidad es bidireccional, entonces la información se suministrará a una herramienta de reingeniería que intentará reestructurar o regenerar el viejo programa.

##### **3.2.2.3. Fases de la Ingeniería Inversa**

El núcleo de la ingeniería inversa es una actividad denominada extracción de abstracción. El ingeniero tiene que evaluar el viejo programa y a partir del código fuente (que no suele estar documentado) tiene que extraer una especificación significativa del procesamiento que se realiza, la interfaz de usuario que se aplica y las estructuras de datos de programa o de base de datos que se utiliza (Ver figura 6).

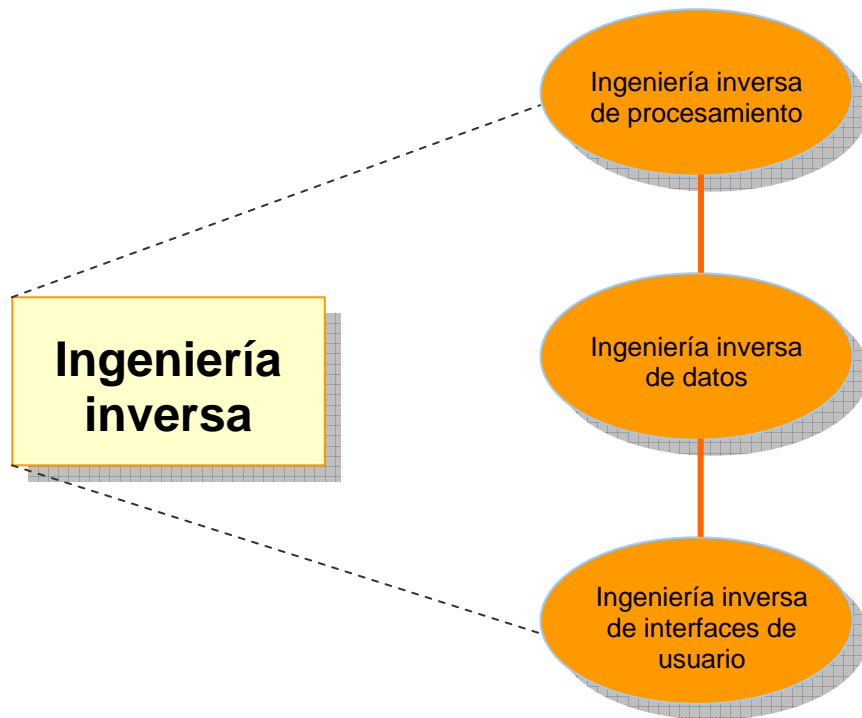


Figura 5. Fases de la Ingeniería Inversa

### **1. Ingeniería inversa para comprender el procesamiento**

La primera actividad real de la ingeniería inversa comienza con un intento de comprender y extraer después abstracciones de procedimientos representadas por el código fuente. Para comprender las abstracciones de procedimientos, se analiza el código en distintos niveles de abstracción: sistema, programa, componente, configuración y sentencia.

La funcionalidad general de todo el sistema de aplicaciones deberá ser algo perfectamente comprendido antes de que tenga lugar un trabajo de ingeniería inversa más detallado. Esto es lo

que establece un contexto para un análisis posterior, y se proporcionan ideas generales acerca de los problemas de interoperabilidad entre aplicaciones dentro del sistema. Cada uno de los programas de que consta el sistema de aplicaciones representará una abstracción funcional con un elevado nivel de detalle. También se creará un diagrama de bloques como representación de la iteración entre estas abstracciones funcionales. Cada uno de los componentes efectúa una subfunción, y representa una abstracción definida de procedimientos. En cada componente se crea una narrativa de procesamiento. En algunas situaciones ya existen especificaciones de sistema, programa y componente. Cuando ocurre tal cosa, se revisan las especificaciones para preciar si se ajustan al código existente.

Todo se complica cuando se considera el código que reside en el interior del componente. El ingeniero busca las secciones de código que representan las configuraciones genéricas de procedimientos. En casi todos los componentes, existe una sección de código que prepara los datos para su procesamiento (dentro del componente), una sección diferente de código que efectúa el procesamiento y otra sección de código que prepara los resultados del procesamiento para exportarlos de ese componente. En el interior de cada una de estas secciones, se encuentran configuraciones más pequeñas.

Por ejemplo, suele producirse una verificación de los datos y una comprobación de los límites dentro de la sección de código que prepara los datos para su procesamiento. Para los sistemas grandes, la ingeniería inversa suele efectuarse mediante el uso de un enfoque semiautomatizado.

Las herramientas CASE se utilizan para la semántica del código existente. La salida de este proceso se pasa entonces a unas herramientas de reestructuración y de ingeniería directa que completarán el proceso de reingeniería.

## **2. Ingeniería inversa para comprender los datos**

La ingeniería inversa de datos suele producirse a diferentes niveles de abstracción<sup>6</sup>. En el nivel de programa, es frecuente que sea preciso realizar una ingeniería inversa de las estructuras de datos internas del programa, como parte del esfuerzo general de la reingeniería. En el nivel del sistema, es frecuente que se efectúe una reingeniería de las estructuras globales de datos (por ejemplo: archivos, bases de datos) para ajustarlas a los paradigmas nuevos de gestión de bases de datos (por ejemplo, la transferencia de archivos planos a unos sistemas de bases de datos relacionales u orientados a objetos).

---

<sup>6</sup> Distintos niveles que posibilitan la resolución de un problema sin tener en cuenta los detalles por debajo de cierto nivel.



La ingeniería inversa de las estructuras de datos globales actuales establece el escenario para la introducción de una nueva base de datos que abarque todo el sistema. Esta comprende:

**a. Estructuras de datos internas.**

Las técnicas de ingeniería inversa para datos de programa internos se centran en la definición de clases de objetos. Esto se logra examinando el código del programa en un intento de agrupar variables de programa que estén relacionadas. En muchos casos, la organización de datos en el seno del código identifica los tipos abstractos de datos. Por ejemplo, las estructuras de registros, los archivos, las listas y otras estructuras de datos que suelen proporcionar una indicación inicial de las clases.

Para la ingeniería inversa de clases, se podría sugerir el enfoque siguiente [3]:

- (i) Identificación de los indicadores y estructuras de datos locales dentro del programa que registran información importante acerca de las estructuras de datos globales (por ejemplo, archivos o bases de datos).
- (ii) Definición de la relación entre indicadores y estructuras de datos locales y las estructuras de datos globales. Por ejemplo, se podrá activar un indicador cuando un archivo

esté vacío; una estructura de datos local podrá servir como memoria intermedia de los cien últimos registros recogidos para una base de datos central.

(iii) Para toda variable (dentro de un programa) que represente una matriz o archivo, la construcción de un listado de todas las variables que tengan una relación lógica con ella.

Estos pasos hacen posible que el ingeniero del software identifique las clases del programa que interactúan con las estructuras de datos globales.

#### **b. Estructuras de bases de datos.**

Independientemente de su organización lógica y de su estructura física, las bases de datos permiten definir objetos de datos, y apoyan los métodos de establecer relaciones entre objetos.

Por tanto, la reingeniería de un esquema de bases de datos para formar otro exige comprender los objetos ya existentes y sus relaciones.

Para definir el modelo de datos existente como precursor para una reingeniería que producirá un nuevo modelo de base de datos se pueden emplear los pasos siguientes [4]:

(i) Construcción de un modelo de objetos inicial

Las claves definidas como parte del modelo se podrán conseguir mediante la revisión de registros de una base de datos de archivos planos o de tablas de un esquema relacional. Los elementos de esos registros o tablas pasarán a ser atributos de una clase,

(ii) Determinación de los candidatos a claves.

Los atributos se examinan para determinar si se van a utilizar o no para señalar a otro registro o tabla. Aquellos que sirvan como punteros pasarán a ser candidatos a claves.

(iii) Refinamiento de las clases provisionales.

Se determina si ciertas clases similares pueden o no combinarse dentro de una Única clase.

(iv) Definición de las generalizaciones

Para determinar si se debe o no construir una jerarquía de clases con una clase de generalización como precursor de todos sus descendentes se examinan las clases que pueden tener muchos atributos similares.

(v) Descubrimiento de las asociaciones

Mediante el uso de técnicas análogas al enfoque de CRC<sup>7</sup> se establecen las asociaciones entre clases.

---

<sup>7</sup> CRC es el modelado de clases – responsabilidades – colaboración

Una vez que se conoce la información definida en los pasos anteriores, se pueden aplicar una serie de transformaciones para hacer corresponder la estructura de la vieja base de datos con una nueva estructura de base de datos.

### **3. Ingeniería inversa de interfaces de usuario**

Las IGUs<sup>8</sup> sofisticadas se van volviendo de rigor para los productos basados en computadoras y para los sistemas de todo tipo. Por tanto el nuevo desarrollo de interfaces de usuario ha pasado a ser uno de los tipos más comunes de las actividades de reingeniería. Ahora bien, antes de que se pueda reconstruir una interfaz de usuario, deberá tener lugar una actividad de ingeniería inversa.

¿Cómo puedo entender el funcionamiento de la interfaz de usuario existente?

Para comprender totalmente una interfaz de usuario ya existente (IU), es preciso especificar la estructura y comportamiento de la interfaz. Se sugieren tres preguntas básicas a las cuales hay que responder cuando comienza la ingeniería inversa de la IU [11]:

- ¿Cuáles son las acciones básicas que deberá procesar la interfaz, por ejemplo, acciones de teclado y clic de ratón?

---

<sup>8</sup> IGUs o IUs: Interfaces Gráficas de Usuario.

- ¿Cuál es la descripción compacta de la respuesta de comportamiento del sistema a estas acciones?
- ¿Qué queremos decir con «sustitución», o más exactamente, qué concepto de equivalencia de interfaces es relevante en este caso?

Es importante indicar que una IGU de sustitución puede que no refleje la interfaz antigua de forma exacta (de hecho, puede ser totalmente diferente). Con frecuencia, merece la pena desarrollar metáforas<sup>9</sup> de interacción nuevas. Por ejemplo, una solicitud de IU antigua en la que un usuario proporcione un superior (del 1 a 10) para encoger o agrandar una imagen gráfica. Es posible que una IGU diseñada utilice una barra de imágenes y un ratón para realizar la misma función.

#### **3.2.2.4. Áreas en la Ingeniería Inversa**

##### **1. Redocumentación:**

Es la creación o revisión de una representación equivalente semánticamente dentro del mismo nivel de abstracción relativo. Los formularios resultantes de representación son usualmente consideradas vistas alternativas (por ejemplo: diagrama de flujo, estructura de datos, flujo de control).

---

<sup>9</sup> Buscan una similitud entre la tarea a realizar y la realidad cotidiana en las IUs.

## **2. Recuperación de diseño**

Es un subconjunto de la ingeniería inversa, en el cual, aparte de las observaciones del sistema, se añaden conocimientos sobre su dominio de aplicación, información externa, y procesos deductivos con el objeto de identificar abstracciones significativas a un mayor nivel.

### **3.2.2.5. Inconvenientes de la Ingeniería Inversa**

- La única fuente confiable es el código que está poco (o nada) documentado y se ha perdido contacto con los diseñadores y/o programadores.
- Las herramientas CASE proveen buen soporte para análisis estático pero limitado para análisis dinámico.

### **3.2.2.6. Herramientas CASE para la Ingeniería Inversa**

Ya existen algunas herramientas de este tipo. Su evolución marcará notables mejoras en la obtención de los diseños a partir del código ya existente (ingeniería inversa) y la regeneración del mismo, una vez optimizado el diseño (ingeniería directa).

Ejemplo de alguno de ellos son:

- Rational Rose
- Telelogic Tau
- Enterprise Architect

- ReWeb, WARE, WANDA (Dominio Web)
- VAQUISTA
- TERESA
- Altova Umodel (Java, C# a UML)
- Imagix 4D (C/C++ a UML)

### **3.3. Definición de términos Básicos**

#### **3.3.1. Ingeniería Inversa**

La Ingeniería Inversa se ocupa de estudiar un sistema de información en el orden inverso establecido en el ciclo de vida habitual; esto es, partiendo del código fuente, se trata de identificar los componentes del sistema y las relaciones existentes entre ellos. Hasta su llegada, el ciclo de vida del software era, en teoría, en una sola dirección; ahora, se puede hablar de dos direcciones: forward o hacia adelante, que es la tradicional, y reverse o hacia atrás, que es la de la Ingeniería Inversa.

#### **3.3.2. Reingeniería**

Es el examen y alteración de un sistema para reconstruirlo de una nueva forma y la subsiguiente implementación de esta nueva forma.

#### **3.3.3. UML**

El UML (Lenguaje Unificado para la Construcción de Modelos) se define como un “lenguaje” que permite especificar, visualizar y construir los artefactos de los sistemas de software” [5]. Es un sistema rotacional

(que, entre otras cosas, incluye el significado de sus notaciones) destinado a los sistemas de modelado que utilizan conceptos orientados a objetos.

#### **3.3.4. Programación Orientada a Objetos**

La Programación Orientada a Objetos (POO u OOP según siglas en inglés) es un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan *estado* (es decir, datos), *comportamiento* (esto es, procedimientos o *métodos*) e identidad (propiedad del objeto que lo diferencia del resto). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar. De esta forma, un objeto contiene toda la información, (los denominados atributos) que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases (e incluso entre objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos). A su vez, dispone de mecanismos de interacción (los llamados métodos) que favorecen la comunicación entre objetos (de una misma clase o de distintas), y en consecuencia, el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separan (ni deben separarse) información (datos) y procesamiento (métodos).



## CAPITULO IV

### 4. METODOLOGIA PROPUESTA

Debido a la inexistencia de una metodología definida para la ingeniería inversa de software y en base a investigaciones realizadas sobre este tema, se propone una metodología que ayude al proceso de ingeniería inversa en la recuperación de la documentación.

Mencionando además que la metodología esta enfocada a la parte conceptual, la cual será la que seguirá al proceso especificado en la siguiente figura (ver figura 6).

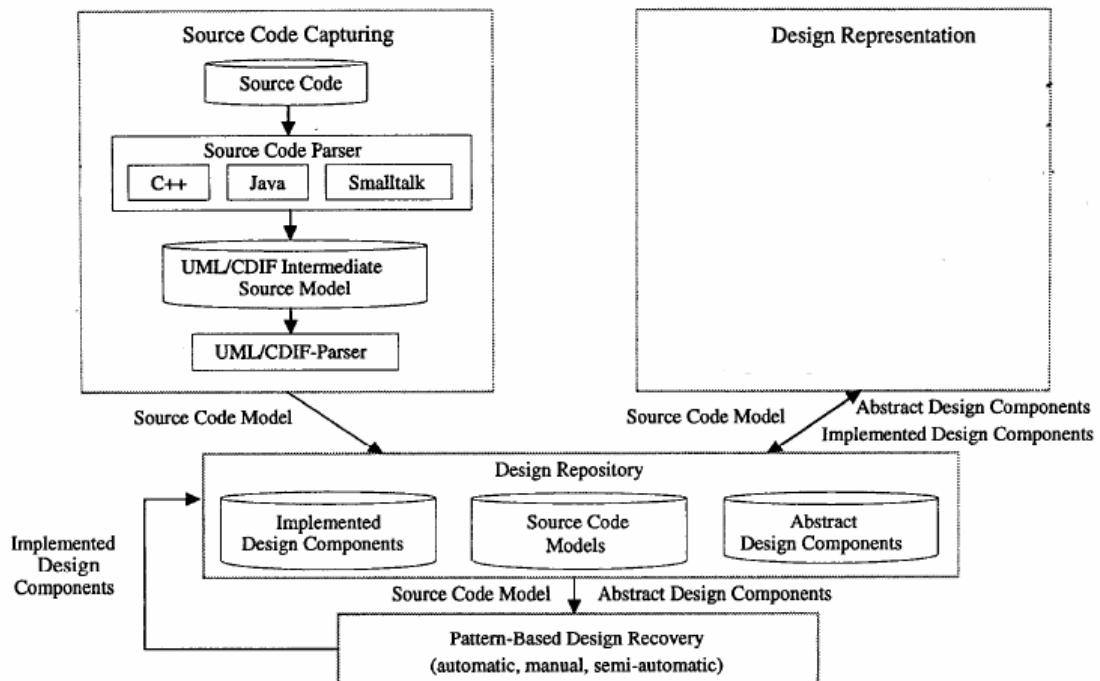


Figura 6. Del código fuente hacia el modelo conceptual

En la figura 6, mediante un parser<sup>10</sup> el código fuente es sometido a un análisis sintáctico a través del cual se obtiene un modelo de código fuente intermedio, el cual es comparado con un conjunto de modelos (usando para esto patrones de diseño) y dando como resultado un modelo conceptual.

La metodología propuesta esta compuesta de las siguientes etapas:

1. Estudio del sistema existente

Una técnica que ayudara al entendimiento del sistema será: la entrevista.

2. Recuperación arquitectónica

Para llevar acabo esta etapa se utilizará el enfoque basado en 5 vistas, cada vista servirá para recuperar cierta información de diseño. Entre las tareas de esta fase tenemos:

- 2.1. Identificación de las tareas funcionales del sistema.

- Diagrama de casos de uso

- 2.2. Recuperación de la vista de diseño

- Utilización de una herramienta CASE

- 2.3. Representación de la interacción entre los objetos

- Diagrama de Secuencia y Colaboración

- 2.4. Identificación de componentes de software

- Diagrama de componentes

- 2.5. Identificación de componentes para el despliegue

- Diagrama de despliegue

3. Documentación de tareas y funciones del sistema.

---

<sup>10</sup> es un programa que reconoce si una o varias cadenas de caracteres forman parte de un determinado lenguaje

Como caso de estudio, la metodología se aplica a un sistema de reservación de hoteles y mantenimiento de agencias de viaje (TravelPlus).

#### **4.1. Estudio del sistema existente**

Lo que se pretende en esta etapa, es lograr un entendimiento general de la funcionalidad del sistema, antes de que éste sea sometido al proceso de ingeniería inversa.

El equipo encargado del proceso de ingeniería inversa deberá contactarse con analistas funcionales, arquitectos de sistema, ingenieros de software y/o con los usuarios, tal que puedan explicar las tareas del sistema (funcionamiento, desarrollo, implementación, etc.).

Como técnica para poder comprender el funcionamiento del sistema hemos seleccionado las entrevistas.

#### **Tipos de entrevista:**

- Entrevistas cerradas: se buscan respuestas a un conjunto de preguntas predefinidas.
- Entrevistas abiertas: no hay agenda predefinida, se realiza un dialogo de manera abierta

Para el caso de estudio luego de entrevistar a los implicados se puede definir lo que hace el sistema a grandes rasgos.

*TravelSys es una empresa que ofrece el inventario de una serie de hoteles a nivel del mundo a agencias de viajes, a través de su sistema TravelPlus. El negocio esta en la comisión que TravelSys cobra a cada agencia por cada reservación que estos realizan.*

*Los viajeros o incluso otras agencias de viajes pueden hacer reservaciones directamente en línea, por teléfono, o enviando una solicitud por correo electrónico.*

*El operador buscara el hotel conveniente haciendo una búsqueda con las características que el viajero desee.*

*Se ingresara los datos de los ocupantes de las habitaciones y finalmente se generará un código de reservación.*

*El Administrador del sistema tendrá la capacidad de poder crear nuevas agencias que quieran trabajar con TravelSys y cancelarlos, también podrá crear usuarios y hacer mantenimiento.*

Las interfaces pueden ayudar mucho al entendimiento de las tareas, a continuación se muestran alguna de ellas:



**NUEVA AGENCIA.**

\*Indica que los campos

<b>Nombres:</b> *	<input type="text"/>	<b>Fax:</b>	<input type="text"/>
<b>Teléfono:</b> *	<input type="text"/>	<b>País:</b>	<input type="text"/>
<b>Dirección:</b> *	<input type="text"/>	<b>(Seleccione)</b>	<input type="text"/>
<b>Ciudad:</b> *	<input type="text"/>	<b>Código ZIP:</b> *	<input type="text"/>
<b>Dirección Web:</b>	<input type="text"/>	<b>E-mail</b>	<input type="text"/>
<b>Banco:</b> *	<input type="text"/>	<b>Número de Cuenta:</b> *	<input type="text"/>
<b>Comisión de Vendedor:</b> *	<input type="text"/>	<b>Comisión de Agencia:</b> *	<input type="text"/>
<b>Días de Anticipación:</b> *	<input type="text"/>	<b>Plazo de Pago(días):</b> *	<input type="text"/>
<b>Impuesto:</b> *	<input type="text"/>	<b>Línea de crédito:</b> *	<input type="text"/>
<b>Estado:</b>	<input type="text"/>	<b>Imagen de Cabecera:</b>	<input type="text"/>
<b>Habilitado</b>	<input type="text"/>	<input type="button" value="Examinar..."/>	

Figura 9. Pantalla de Ingreso de una nueva Agencia

**Editando Agencia**

\*Indica que los campos

**Código Zip:** Ingrese un Código de Zip.  
**Banco:** Ingrese un nombre de Banco por favor.  
**Plazo de Pago:** Ingrese una cantidad mayor o igual a 2.

<b>Nombres:</b> *	<input type="text"/>	<b>Fax:</b>	<input type="text"/>
<b>Teléfono:</b> *	<input type="text"/>	<b>País:</b>	<input type="text"/>
<b>Dirección:</b> *	<input type="text"/>	<b>(Seleccione)</b>	<input type="text"/>
<b>Ciudad:</b> *	<input type="text"/>	<b>Código ZIP:</b> *	<input type="text"/>
<b>Dirección Web:</b>	<input type="text"/>	<b>E-mail</b>	<input type="text"/>
<b>Banco:</b> *	<input type="text"/>	<b>Número de Cuenta:</b> *	<input type="text"/>
<b>Comisión de Vendedor:</b> *	<input type="text"/>	<b>Comisión de Agencia:</b> *	<input type="text"/>
<b>Días de Anticipación:</b> *	<input type="text"/>	<b>Plazo de Pago(días):</b> *	<input type="text"/>
<b>Impuesto:</b> *	<input type="text"/>	<b>Línea de crédito:</b> *	<input type="text"/>
<b>Estado:</b>	<input type="text"/>	<b>Imagen de Cabecera:</b>	<input type="text"/>
<b>Habilitado</b>	<input type="text"/>	<input type="button" value="Examinar..."/>	

Figura 10. Pantalla de Modificación de datos de una Agencia

**BUSQUEDA DE HOTELES**

**Ubicación**  
 Continente:   
 País:   
 Estado:   
 Ciudad:

**Selección de Fechas**  
 Ingreso:    
 Salida:

**Información de Viajero**  
**Cuartos y Huespedes**  
 Cuartos:   
 Adultos:  Niños:   
   
 Edad:   
 Hotel:   
 Estrellas:   
 Cadena:   
 Precio Max:  ☒ Disponible

**Buscar**

Figura 11. Pantalla de Búsqueda de Hoteles

**BUSQUEDA DE HOTELES**

**Ubicación**  
 Continente:   
 País:   
 Estado:   
 Ciudad:

**Selección de Fechas**  
 Ingreso:    
 Salida:

**Información de Viajero**  
**Cuartos y Huespedes**  
 Cuartos:   
 Adultos:  Niños:   
   
 Edad:   
 Hotel:   
 Estrellas:   
 Cadena:   
 Precio Max:  ☒ Disponible

**Buscar**

**1 2 3 ...**  
 Resultados encontrados: 49 hoteles.

**Broadway Hotel & Hostel - DEMO**

**New York,NY,US**  
 DEMO - Hosting over 300 beds and select private rooms, the Broadway Hotel & Hostel provides comfortable accommodations to meet any travel budget. Warm color tones and an Asian-themed accent throughout the hotel create an exciting environment.

**\$127**  
Precio promedio mínimo

[más info](#)

Tipo de cuarto	Estado	Costo prom. x noche
8 Bed Dorm Shared Bath	True	\$126.95
Fecha:	09/19/2007	09/20/2007
Disponibilidad:	Disponible	Disponible
Duermen 2	\$126.95	\$126.95

Tipo de cuarto	Estado	Costo prom. x noche
6 Bed Dorm Shared Bath	True	\$135.00
Fecha:	09/19/2007	09/20/2007
Disponibilidad:	Disponible	Disponible
Duermen 2	\$135.00	\$135.00

Tipo de cuarto	Estado	Costo prom. x noche
4 Bed Dorm Shared Bath	True	\$147.08
Fecha:	09/19/2007	09/20/2007
Disponibilidad:	Disponible	Disponible
Duermen 2	\$147.08	\$147.08

Tipo de cuarto	Estado	Costo prom. x noche
Standard Double Private Bath	True	\$277.88
Fecha:	09/19/2007	09/20/2007
Disponibilidad:	Disponible	Disponible
Duermen 2	\$277.88	\$277.88


**Continental Hostel - DEMO**

**New York,NY,US**  
 DEMO - Continental Hostel is a property conveniently located to major attractions and

**\$100**  
Precio

Figura 12. Pantalla de Hoteles encontrados

W - Times Square - DEMO



DEMO - 1567 BROADWAY & 47TH STREET  
New York NY 10036  
USA

Tipo de habitación	Disponibilidad	Precio Promedio Por Noche	
Wonderful Room	Disponible	988.86	<a href="#">Reservar</a>

Fecha:	09/20/2007	09/21/2007
Disponibilidad:	Disponible	Disponible
Duermen 3	1162.65	815.06

W - Times Square - DEMO - Descripción

General information: Welcome to W New York - Times Square, an oasis of calm in the midst of the flurry of a revitalized Times Square. From the moment you step inside, you will be transported to a Zen-like environment. Quiet colors, calming textures and a dramatic glass-encased waterfall that soothes the senses. Water runs overhead, streaming down on open sides, enabling you to literally "walk on water" as you enter the elevator that whisks you to Reception. Location: W New York—Times Square is in the heart of the theater district, with museums a few blocks north and sultry downtown a quick subway ride away. Area details: The hotel is nearby Broadway theaters, Fifth Avenue Shopping and international businesses. Recreation: See and be seen. Tucked beneath the hotel you will discover a pulsating destination that defines the essence of New York City nightlife. From the mind of Rande Gerber, THE WHISKEY is a 7,500-square foot bar/screening room and the ultimate nightlife experience. Daring and stimulating, THE WHISKEY features an array of New York's hottest DJs spinning an eclectic mix of lively beats, and a state-of-the-art screening room. The hotel facilities include Sweat Fitness Cetner, Away Spa, W The Store, Meeting Facilities, Lounge, Blue Fin Restaurant, W Living Room. Rooms: W New York - Times Square features 507 sleek ultra-modern guest rooms, including 43 suites, with all the trimmings. Modern and linear, each room is a marvel in design and pillar of comfort. Luminous amber-colored resin cubes serve as side tables to the famously comfortable W signature bed. A floating glass desk looks out onto the breathtaking New York City skyline. Sexy bathrooms with round white sinks suspend as if in mid-air, and open showers or luxurious tubs reside behind white translucent sliding screen doors.

Figura 13. Pantalla de Detalle de Hotel

RESUMEN DE VIAJE

Hotel:  
**Broadway Hotel & Hostel - DEMO**

Dirección:  
**DEMO - 230 West 101 Street New York  
New York 10025**

Habitaciones: 2

Tipo de Habitación:  
**8 Bed Dorm Shared Bath**

Adultos/Niños: 3/1

Inicio: 9/19/2007

Final: 9/21/2007

Costo del Hotel: \$253.90

Impuest. Hotel: \$40.90

Impuest. Agencia: \$53.06

**Total: \$347.86**

Por favor ingrese un contacto principal por cada habitación.

Habitación 1

Primer Nombre

Apellido Paterno

Apellido Materno

Preferencias de Habitación

☒ Duermen 2

Requerimientos Especiales

Habitación 2

Primer Nombre

Apellido Paterno

Apellido Materno

Preferencias de Habitación

☒ Duermen 2

Requerimientos Especiales

Continuar

Figura 14. Pantalla de Ingreso de datos de la reservación

63



<b>RESUMEN DE VIAJE</b>	<b>Gracias por realizar su reservación en WEBTRAVELSYS.</b> <b>Numero de Reservación: 5226693</b>
Hotel: <b>Broadway Hotel &amp; Hostel - DEMO</b> Dirección: <b>DEMO - 230 West 101 Street New York New York 10025</b> Habitaciones: 2 Tipo de Habitación: <b>8 Bed Dorm Shared Bath</b> Adultos/Niños: 3/1 Inicio: 9/19/2007 Final: 9/21/2007 Costo del Hotel: \$253.90 Impuest. Hotel: \$40.90 Impuest. Agencia: \$53.06 <b>Total: \$347.86</b>	
	<a href="#">Continuar</a>

Figura 15. Pantalla de constancia de reservación

## 4.2. Recuperación Arquitectónica

El uso de la ingeniería inversa para reconstruir los aspectos arquitectónicos del software puede referenciarse como redocumentación estructural o recuperación arquitectónica. Como resultado, se deriva la visión global del sistema de interés y se pueden recapturar algunos aspectos de su diseño arquitectónico.

Se presenta un enfoque de arquitectura de sistemas basado en vistas (ver figura 16). Estas vistas muestran los diferentes aspectos del sistema que son modelados.

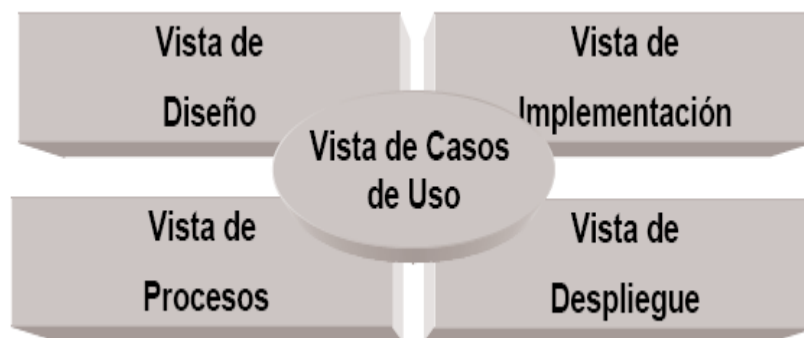


Figura 16. Descripción del modelo desde 5 vistas

Cada una de estas vistas servirá para realizar una tarea específica, para la descripción del contenido de cada una de estas vistas usaremos los diagramas. El estándar que hemos adoptado es el UML (Lenguaje de Modelamiento Unificado) ya que proporciona un amplio conjunto de diagramas que normalmente se usan en pequeños subconjuntos para poder representar las cinco vistas principales de la arquitectura de un sistema. En

el cuadro adjunto relacionamos las tareas, la vista y los diagramas que utilizaremos en cada uno de ellos.

Tarea	Vista	Diagrama
1. Identificación de requerimientos funcionales	Casos de uso	Casos de uso
2. Recuperación del diagrama de clases	Diseño	Clases
3. Representación de la interacción entre los objetos	Procesos	Secuencia y colaboración
4. Identificación de componentes de software	Implementación	Componentes
5. Identificación de componentes para el despliegue	Despliegue	Despliegue

### **Tarea 1: Identificación de requerimientos funcionales**

Luego de conocer en mayor detalle el sistema, los *diagramas de caso de uso* nos ayudaran a realizar esta tarea. En la ingeniería directa, el análisis se enfoca en la funcionalidad y luego el diseño agrega los detalles específicos de la implementación y los aspectos relacionados con los requerimientos no funcionales. En la ingeniería inversa, todos los detalles de diseño e implementaron ya están definidos, de esta manera el limite entre el análisis y el diseño es bastante difuso.

En la ingeniería inversa, los casos de uso permiten descubrir y describir, en un alto nivel, que hace el sistema desde el punto de vista del usuario. En esta actividad, *no interesa como hará algo el sistema*.

### **Forma de aplicación de la ingeniería inversa en casos de uso:**

- a. Identificar cada actor que interactúa con el sistema.
- b. Para cada actor, considerar la manera en la cual el actor interactúa con el sistema, cambia el estado del sistema o su medio ambiente, o responde a algún evento.
- c. Trazar el flujo de eventos en el sistema ejecutable relativo a cada actor. Iniciar con flujos primarios y después considerar rutas alternativas.
- d. Agrupar flujos relacionados declarando su correspondiente caso de uso.
- e. Proporcionar estos actores y casos de uso en un diagrama de casos de uso, y establecer sus relaciones.

Entre los requisitos funcionales pertenecientes al sistema TravelPlus se identificaron:

#### **Actores**

Los actores del sistema fueron identificados como:

- Administrador: Es la persona que utiliza la parte administrativa y de mantenimiento de tablas como Agencia y Usuarios.

- Usuario: Es la persona que utiliza el sistema de reservaciones.
- Cliente: Es la persona que solicita una reservación de habitaciones.

### **Casos de Uso**

Basados en los actores y la funcionalidad del sistema fueron identificados los siguientes paquetes que agrupan casos de uso relacionados:

- Gestión de Agencias de Viaje: Agrupa funcionalidades relacionada a la agencias de viaje como son: Ingresar, modificar y eliminar.
- Gestión de usuarios: Agrupa funcionalidades relacionada a la administración de usuarios del sistema como son: Ingresar, modificar y eliminar.
- Reservación de Hoteles: Agrupa funcionalidades relacionados al proceso de reservación las cuales empiezan en la solicitud de reservación por parte del cliente, el registro del mismo, hacer, consultar o cancelar una reservación.

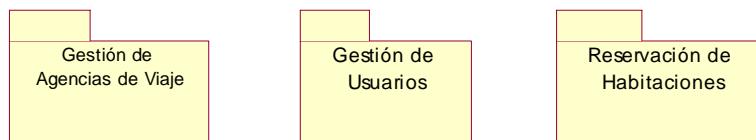


Figura 17. Gestión del Sistema TravelPlus

Los casos de uso identificados:

- *Ingresar Agencia:* este caso de uso es iniciado por el administrador, consiste en crear una nueva agencia de viaje al sistema.
- *Modificar Agencia:* este caso de uso es iniciado por el administrador, consiste en modificar datos de una agencia de viaje para lo cual primero debe hacer una búsqueda de la agencia.
- *Eliminar Agencia:* este caso de uso es iniciado por el administrador, consiste en cambiar el estado de una agencia de viaje siempre en cuando esta no cumpla con el pago para lo cual primero debe hacer una búsqueda de la agencia.
- *Búsqueda Agencia:* Proporciona la capacidad de buscar una agencia.
- *Ingresar Usuario:* este caso de uso es iniciado por el administrador, consiste en crear un nuevo usuario que interactuara con el sistema.

- *Modificar Usuario:* este caso de uso es iniciado por el administrador, consiste en modificar los datos de un usuario del sistema para lo cual primero debe hacer una búsqueda del usuario.
- *Eliminar Usuario:* este caso de uso es iniciado por el administrador, consiste en eliminar un usuario del sistema para lo cual primero debe hacer una búsqueda del usuario.
- *Búsqueda Usuario:* Proporciona la capacidad de buscar un usuario.
- *Registrar Cliente:* este caso es iniciado por el usuario, consiste en registrar a un nuevo cliente al sistema, previo a esto debe registrar la tarjeta de crédito del cliente.
- *Registrar Tarjeta:* Proporciona la capacidad de registrar la tarjeta de crédito del cliente.
- *Solicitar Reservación:* este caso es iniciado por el cliente, consiste en solicitar una reservación de habitaciones, entre los datos de la reservación tenemos: fecha, numero de cuartos, numero de niños por cuarto, etc.

- *Hacer Reservación:* este caso es iniciado por el usuario, consiste en registrar la reservación en el sistema de acuerdo a la solicitud del cliente.
- *Consultar Información:* este caso de uso es iniciado por el usuario, consiste en consultar información relacionada a la reservación.
- *Cancelar Reservación:* este caso de uso es iniciado por el usuario, consiste en cancelar la reservación a petición del cliente.

Todos estos casos de uso han sido implementados en el sistema. Estas funcionalidades identificadas han sido modeladas usando la herramienta case Rational Rose. Los diagramas de casos de uso se muestran a continuación.

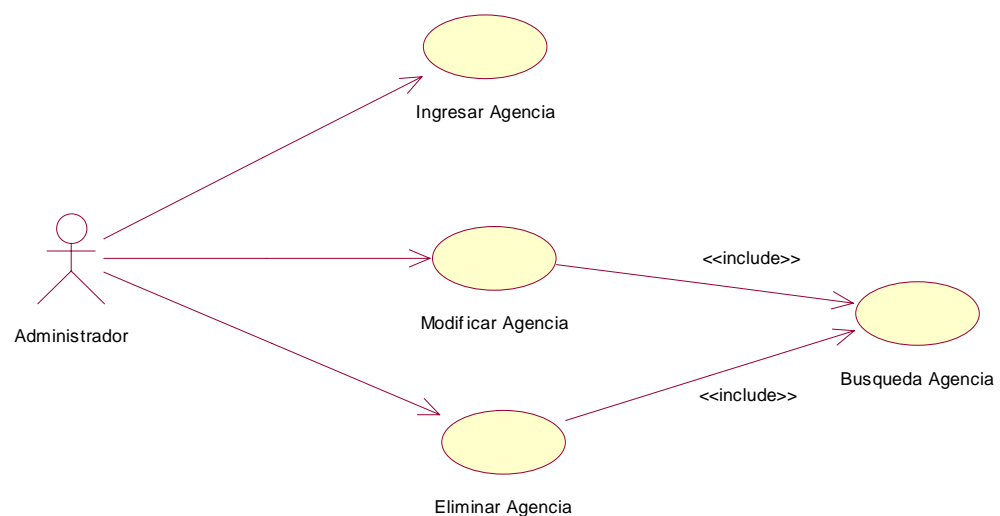


Figura 18. Diagramas de Caso de Uso – Gestión Agencia de Viaje



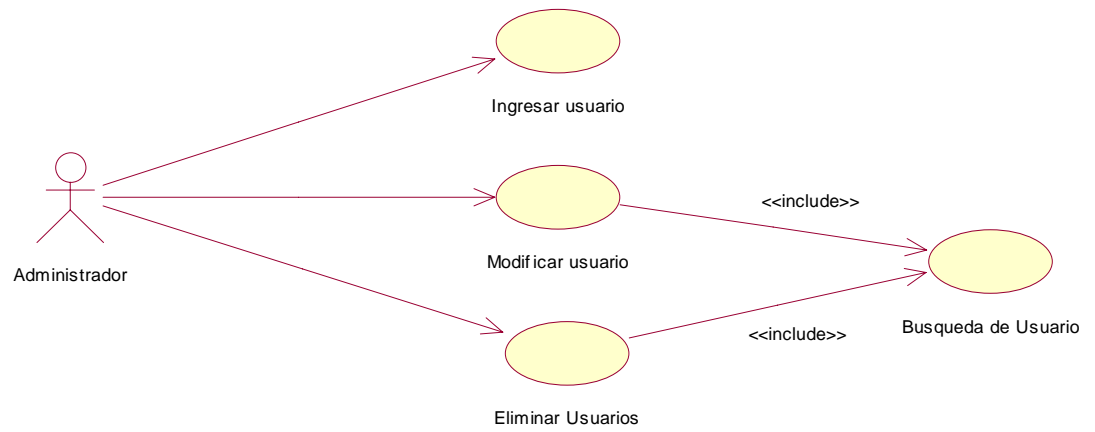


Figura 19. Diagramas de Caso de Uso – Gestión de Usuarios

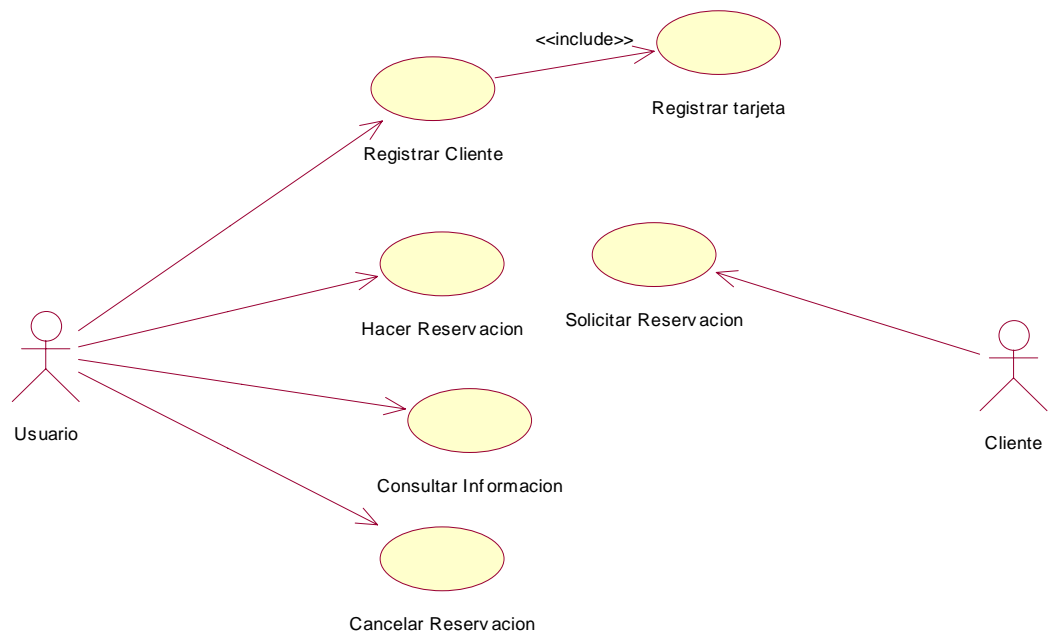


Figura 20. Diagramas de Caso de Uso – Reservación de Habitaciones

## **Tarea2: Recuperación del diagrama de clases**

La *vista de diseño* soporta principalmente los requisitos funcionales del sistema, o sea, los servicios que el sistema debe proporcionar. Es en esta vista donde se obtiene una visión global del funcionamiento del sistema. Se establecen los primeros requisitos funcionales en base a lo que se espera conseguir.

En esta etapa se identifican, dentro del código, las clases que realizan o soportan las acciones descritas en los casos de uso. Para la recuperación de los diagramas de clases haremos uso de una herramienta CASE.

## **Consideraciones para la selección de una herramienta CASE**

Al seleccionar una herramienta CASE se debe verificar que contemple las siguientes funciones específicas:

- Soporte como mínimo a las fases de análisis y diseño de sistemas, utilizando los modelos de representación de procesos, datos, estado-evento, y estructura.
- Contar con un editor fácil de usar y consistente para la creación y mantenimiento de diagramas.
- Diccionario de datos abierto e integrado, con herramientas para el mantenimiento de su contenido.
- Permitir la importación y exportación de datos.

- Características de trabajo en grupo, como seguridad de acceso y modificación a los datos, estandarización de metodologías para un proyecto, y definición de usuarios, claves, privilegios y permisos.
- Generación automática de la documentación técnica del sistema a través de diversas opciones de reportes.

Otros aspectos a considerar para la elección de herramientas son:

- Costo
- Lenguajes que soportan
- Sistemas operativos bajo los cuales funcionan
- Requerimientos del sistema
- Facilidad de uso
- Buen rendimiento

En la siguiente tabla se presenta algunas herramientas CASE en el mercado que realizan el proceso de ingeniería inversa.

Herramienta CASE	Lenguajes	Sistema Operativo	Costo \$
WithClass	C++, Java, Delphi, VB, IDL, Perl, PHP, C# y VB.net	Windows	495
Enterprise architect	incluyendo C++, C#, Java, Delphi, VB.Net, Visual Basic y PHP	Windows, Linux	335
Modelmarker	DELPHI / C#	Windows	299
Poseidon for UML	Java	Windows NT/2000/XP/Vista, Linux, Mac OS X	249
MagicDraw UML Professional	J2EE, C#, C++, CORBA	Windows, Unix	499

Las herramientas de ingeniería inversa y progresiva de la próxima generación harán un uso mucho mayor de técnicas de inteligencia artificial, aplicando una base de conocimientos que se a especifica del dominio de la aplicación (esto es, un conjunto de reglas de descomposición que se aplicarían a todos los programas de una cierta zona de aplicación tal como el control de fabricación o la aviónica). El componente de inteligencia artificial asistirá en la descomposición y reconstrucción del sistema, pero seguirá requiriendo una interacción con un ingeniero de software a lo largo del ciclo de la reingeniería.

Por tratarse de un sistema desarrollado en el lenguaje de programación .NET, se utilizará la herramienta Enterprise Architect.

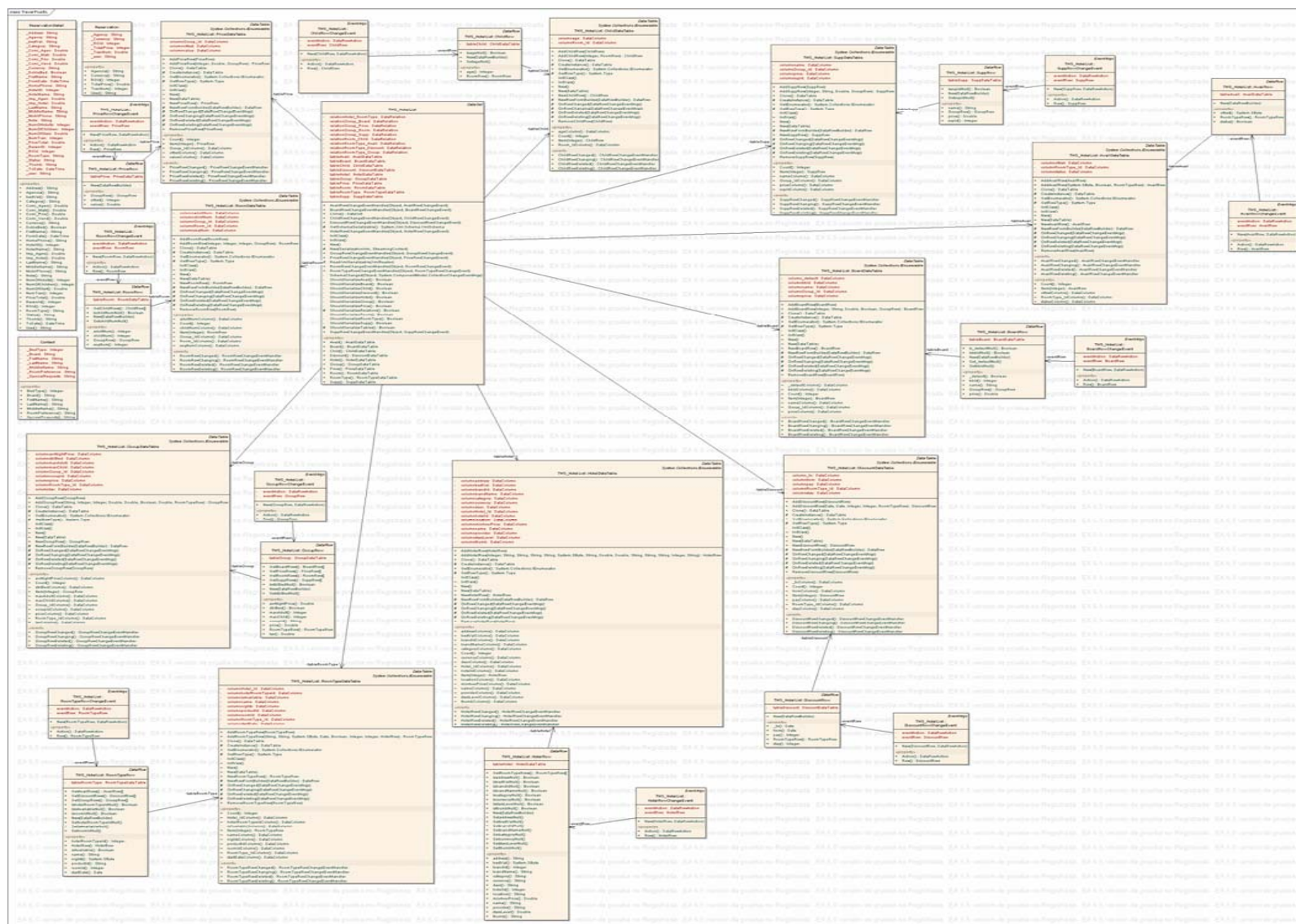


Figura 21. Diagrama de Clases de la capa de negocios



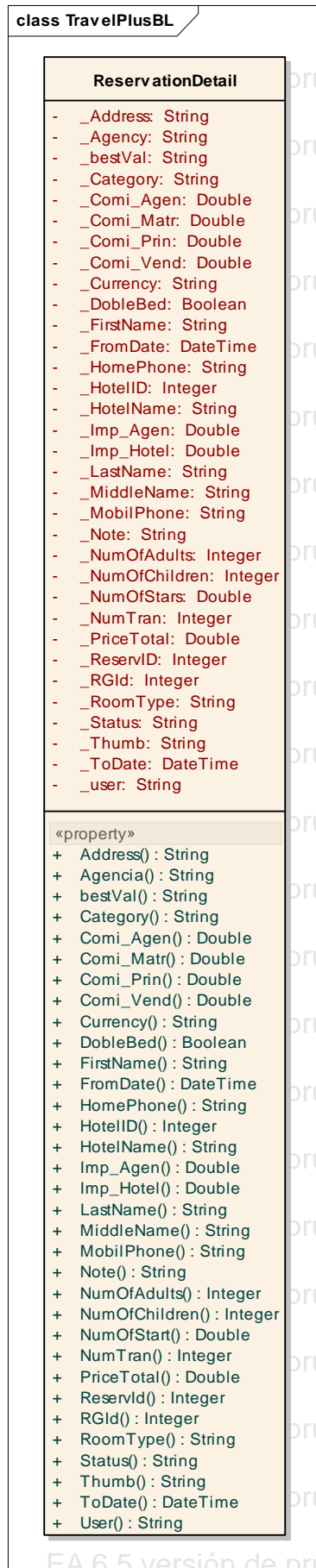


Figura 22. Clase ReservationDetail – Atributos y Métodos

### **Tarea 3: Representación de la interacción entre los objetos**

La *vista de procesos* se utilizara para especificar operaciones son ejecutadas por cada una de las clases identificadas en la vista de diseño y el flujo de colaboración entre ellas. Los casos de uso identificados sirven de guía para encontrar, dentro del código fuente, los objetos que intervienen en el proceso.

Cada caso de uso describe en forma manual, los objetos que intervienen y la interacción que existe entre ellos. La interacción entre objetos se representa mediante diagramas de secuencia, los cuales se construyen en base a los casos de uso, y también con diagramas de colaboración que muestran la interacción entre varios objetos y los enlaces que existen entre ellos.

Para el caso de estudio, se realizó una representación del diagrama de secuencia y de colaboración para el caso de uso Ingresar Agencia. Para representar dichos diagramas utilizaremos la herramienta Rational Rose.



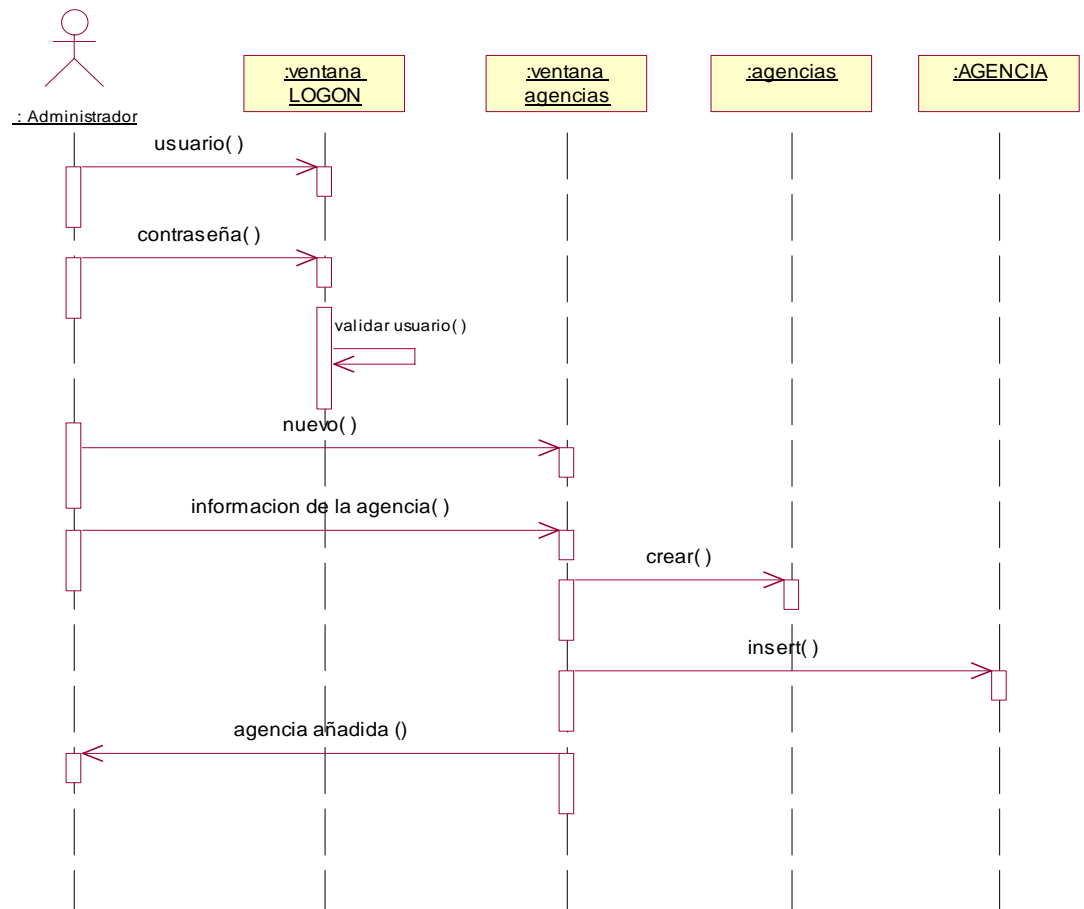


Figura 23. Diagrama de Secuencia para el caso de uso Ingresar Agencia

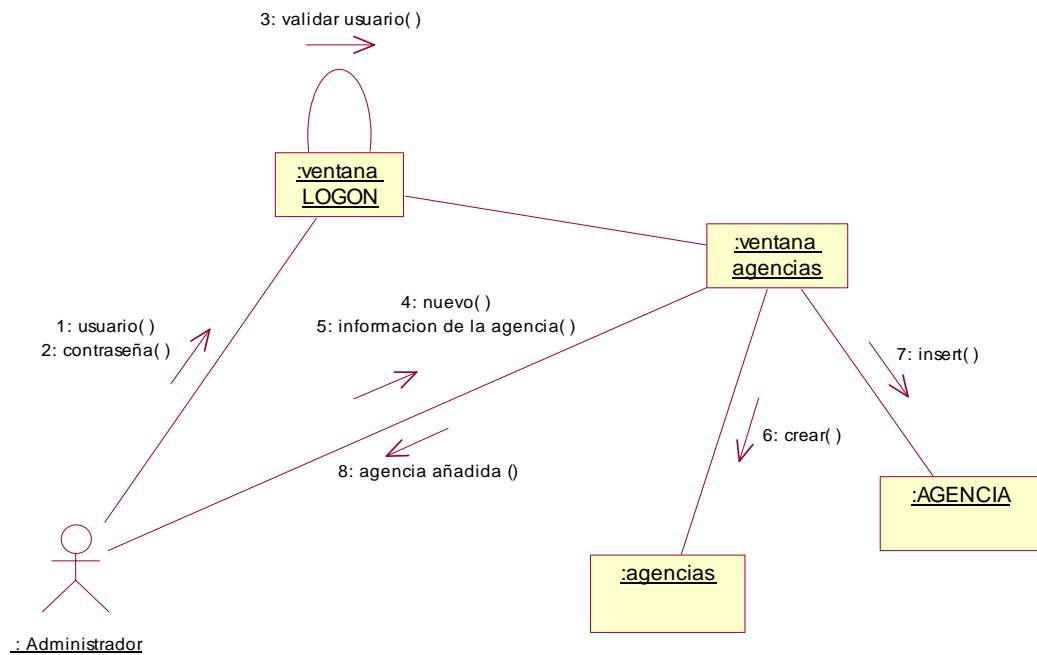


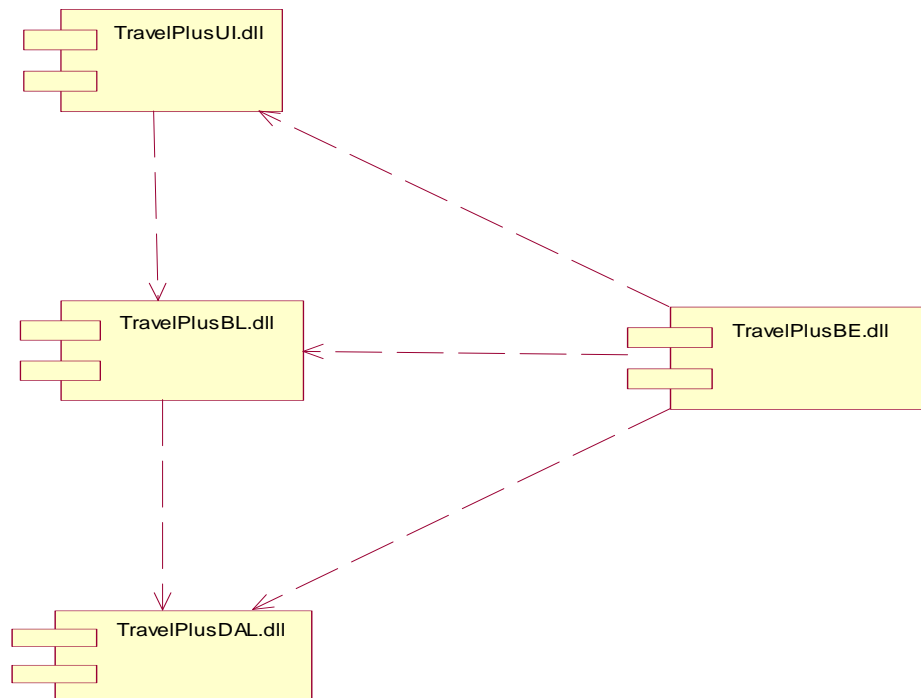
Figura 24. Diagrama de Colaboración para el caso de uso Ingresar Agencia

#### Tarea 4: identificación de componentes de software

La *Vista de implementación* describe la organización estática de los módulos de software (código fuente, componentes, ejecutables) en el ambiente de desarrollo en términos de paquetes, de capas y de la administración de la configuración.

Esta vista puntualiza una correspondencia entre los elementos de la vista de diseño y las entidades físicas que encapsulan su implementación. Asimismo, también se indican las características físicas de los componentes, tales como: memoria requerida, nombre con el que se registra el componente en el sistema, versión y modelo del componente, tiempo de servicio de cada operación, etc., necesarios para realizar la evaluación de la arquitectura.

Los aspectos capturados para el sistema TravelPlus:



### Tarea 5: identificación de componentes para el despliegue

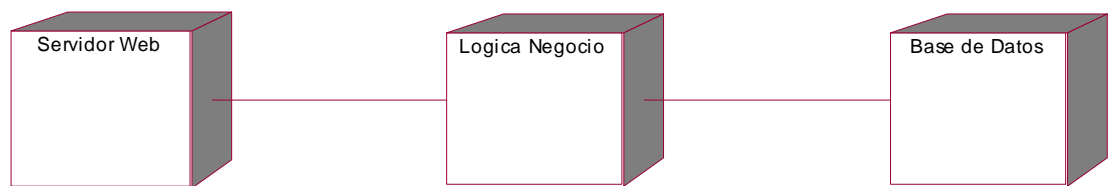
La *vista de despliegue* de un sistema contiene los nodos que forman la topología hardware sobre la que se ejecuta el sistema. Se preocupa principalmente de la distribución, entrega e instalación de las partes que constituyen el sistema. Los aspectos estáticos de esta vista se representan mediante los diagramas de despliegue y los aspectos dinámicos con diagramas de interacción, estados y actividades.

En esta vista se representan las características técnicas de cada uno de los nodos (hosts, en la terminología TCP/IP) del sistema sobre el que se

ejecutara la aplicación, y la asignación de cada componente físico a un nodo.

Describe la puesta en marcha, la instalación y el rendimiento del sistema.

El caso aplicativo, el sistema TravelPlus esta constituido por tres capas, las cuales están representadas en el siguiente diagrama:



#### 4.3. Documentación de tareas y funciones del sistema.

En este documento se especificara la forma de interactuar entre los usuarios del sistema y el sistema bajo estudio. A continuación se define una plantilla que ayudará a la especificación de cada caso de uso.

**Caso de Uso: <Nombre del caso de uso>**

#### Historia de Revisiones

Fecha	Version	Descripcion	Autor
<dd/mm/yyyy>	<1.0>	<Descripción de los cambios>	<xxxx>

## **1. Descripción Breve**

Se debe describir brevemente el propósito del caso de uso. Un simple párrafo será suficiente para la descripción.

## **2. Actores**

Se debe listar a los actores que inician el caso de uso o aquellos que participan en el caso de uso.

## **3. Precondiciones**

Son los hechos que se han de cumplir para que el flujo de evento se pueda llevar a cabo.

## **3. Flujo de Eventos**

Esta relacionado a la secuencia de acciones que se realizan para llevar a cabo el caso de uso. Lo dividiremos en dos tipos de flujos:

### **3.1 Flujo principal**

El caso de uso debe describir que hace el actor y que el sistema da como respuesta. Esta descripción debe ser en frases en forma de dialogo entre el actor y el sistema.

### **3.2 Flujos alternos**

Los flujos alternativos serán descritos en esta parte, previamente referenciado en el flujo básico

A continuación, se documentan los casos de uso del sistema TravelPlus haciendo uso de la plantilla, la cual ayudará en el seguimiento y las modificaciones que se quieran hacer mas adelante. Aclarando que solo se mostrara la documentación de algunos casos de uso.

### **Caso de Uso: Ingresar Agencia**

#### **Historia de Revisiones**

<b>Fecha</b>	<b>Version</b>	<b>Descripcion</b>	<b>Autor</b>
02/09/2007	1.0	Descripción inicial del caso de uso	Jessica Acevedo

#### **1. Descripción Breve**

Este caso de uso consiste en crear una nueva agencia de viaje en el sistema.

**2. Actores:** Administrador

**3. Precondiciones:** El sistema debe validar al administrador

#### **4. Flujo de Eventos**

##### **4.1 Flujo principal**

Este caso de uso inicia cuando el administrador ingresa al sistema. El sistema pide al usuario su nombre y contraseña. El administrador

ingresa nombre y contraseña. El sistema verifica que el nombre y contraseña sean válidos (E-1). El sistema le muestra el listado de agencias. El administrador selecciona la actividad NUEVO. El sistema solicitara datos relacionados a la nueva agencia. El administrador ingresa nombres, teléfono, fax, dirección, ciudad, país, página web (E-2), correo electrónico (E-3), banco, número de cuenta, estado, comisión de agencia, plazo de pago, línea de crédito. El administrador presiona el botón GUARDAR. El sistema guarda los datos de la nueva agencia. El caso de uso inicia de nuevo.

#### **4.2 Flujos alternos**

E-1: El nombre y/o contraseña introducido por el usuario es invalido. El sistema le notifica al administrador. El administrador puede introducir un nombre o contraseña valida o salir del caso de uso.

E-2: El administrador ha introducido una página web que no cumple con el formato. El sistema le notifica al administrador. El administrador puede introducir una página web valida o salir del caso de uso.

E-3: El administrador ha introducido un correo electrónico que no cumple con el formato. El sistema le notifica al administrador. El administrador puede introducir un correo electrónico valido o salir del caso de uso.

## Caso de Uso: Modificar Agencia

### Historia de Revisiones

Fecha	Version	Descripcion	Autor
02/09/2007	1.0	Descripción inicial del caso de uso	Jessica Acevedo

#### 1. Descripción Breve

Este caso de uso permite modificar los datos de una agencia.

#### 2. Actores: Administrador

**3. Precondiciones:** La agencia a modificar debe estar ingresada en el sistema.

#### 4. Flujo de Eventos

##### 4.1 Flujo principal

Este caso de uso inicia cuando el administrador ingresa al sistema. El sistema le pide al usuario su nombre y contraseña. El administrador ingresa nombre y contraseña. El sistema verifica que el nombre y contraseña sean válidos (E-1). El sistema le muestra el listado de agencias. El administrador busca la agencia a modificar y selecciona la opción VER DATOS. El sistema le mostrará los datos registrados en la agencia pero deshabilitados para escribir. El administrador selecciona la opción Editar y empieza a modificar algunos datos como: nombres,



teléfono, fax, dirección, ciudad, país, pagina web (E-2), correo electrónico (E-3), banco, número de cuenta, estado, comisión de agencia, plazo de pago, línea de crédito. El administrador presiona el botón GUARDAR. El sistema guarda los datos modificados de la agencia. El caso de uso inicia de nuevo.

## **4.2 Flujos alternos**

E-1: El nombre y/o contraseña introducido por el usuario es inválido. El sistema notifica al administrador. El administrador puede introducir un nombre o contraseña valida o salir del caso de uso.

E-2: El administrador ha introducido una página web que no cumple con el formato. El sistema le notifica al administrador. El administrador puede introducir una página web válida o salir del caso de uso.

E-3: El administrador ha introducido un correo electrónico que no cumple con el formato. El sistema le notifica al administrador. El administrador puede introducir un correo electrónico válido o salir del caso de uso.

## Caso de Uso: Hacer Reservación

### Historia de Revisiones

Fecha	Version	Descripcion	Autor
02/09/2007	1.0	Descripción inicial del caso de uso	Jessica Acevedo

#### 1. Descripción Breve

Este caso de uso permite que un usuario pueda hacer reservaciones.

#### 2. Actores: Usuario

#### 3. Precondiciones: Las habitaciones deben estar disponibles.

#### 4. Flujo de Eventos

##### 4.1 Flujo principal

Este caso de uso inicia cuando el usuario ingresa al sistema. El sistema pide al usuario su nombre y contraseña. El usuario ingresa nombre y contraseña. El sistema verifica que el nombre y contraseña sean válidos (E-1). El sistema le muestra la pantalla de búsqueda de hoteles. El usuario selecciona continente, país, estado, ciudad, selecciona un rango de fechas (E-2), número de cuartos, por cada cuarto ingresa la cantidad de niños y adultos, selecciona el numero de estrellas del hotel, luego presiona el botón Buscar. El sistema empieza la búsqueda por los criterios ingresados (E-3). El sistema le muestra la lista de hoteles. El

usuario selecciona la mejor oferta y presiona el botón Reservar. El sistema pide que ingrese los datos de un contacto principal por cada habitación. El usuario ingresa datos y presiona el botón continuar. El sistema guarda los datos y le muestra un número de reservación. El caso de uso inicia de nuevo.

#### **4.2 Flujos alternos**

E-1: El nombre y/o contraseña introducido por el usuario es invalido. El sistema le notifica al administrador. El administrador puede introducir un nombre o contraseña valida o salir del caso de uso.

E-2: El administrador ha ingresado una fecha no válida. El sistema notifica al administrador. El administrador puede introducir una fecha válida o salir del caso de uso.

E-3: El sistema no puede encontrar ninguna lista de hoteles. El caso de uso inicia de nuevo.

## **CAPITULO V**

### **5. ESTADO DEL ARTE**

#### **5.1. Ingeniería Inversa en Casos de Uso UML**

Dragan Bojic, Duran Velasevic [7] proponen una técnica basada en el análisis dinámico de un sistema para seleccionados casos de prueba que cubren casos de uso importantes. La teoría de análisis de conceptos formales es aplicada para obtener clasificaciones de elementos del modelo.

Ellos adoptan el enfoque de conducir ingeniería inversa guiado por casos de uso por las siguientes razones:

- Los casos de uso son fácilmente comprensibles por usuarios, como también por desarrolladores, por eso representan un medio de buena comunicación.
- El esfuerzo del diseño se centra en la funcionalidad del antiguo/cambiado sistema antes que el diseño detallado.

Hay varias actividades a realizarse al aplicar la técnica propuesta:

1. Identificar un conjunto de casos de uso para un sistema dado o una parte de esto.
2. Por cada caso de uso, definir uno o más casos de prueba que cubran importantes casos de uso.

3. Recoger la información que perfila para cada caso de prueba.
4. Construir la relación del contexto entre los casos del uso y las entidades cubiertas del código de sistema
5. Construir el enrejado conceptual (por ejemplo, un diagrama acíclico) desde una relación de contexto aplicando análisis formal de conceptos. Informalmente, cada concepto es un par de dos componentes: un conjunto de casos de uso y un conjunto de entidades del código del sistema.
6. Estimar la calidad de descomposición desde una topología de enrejado, y si es necesario modificar el caso de prueba o aplicar alguna purificación de los conceptos relacionados, para eliminar los efectos de interpolación en el código u otros efectos no deseados y repetir actividades necesarios para obtener el enrejado actualizado.
7. Construir el básico modelo UML usando analizador estático.

### Caso Aplicativo

El ejemplo de aplicación es Scribble desde Microsoft Visual C++, este es un MDI aplicación que soporta elementos de dibujo, el cual puede ser guardado y recuperado. Tiene un total de 34 clases y 401 funciones miembros y no miembros. En su investigación ellos definen los siguientes casos de uso que corresponden a los ítems del menú principal:

U1. Trabajando con documentos (especifico con cada U2 o U3)

U2. Creando, abriendo y guardando

U3. Imprimiendo

U4. Trabajando con múltiples ventanas

U5. Dibujando

U6. Configurando

U7. Ayuda

Plantean doce casos de prueba para cubrir los casos de uso.

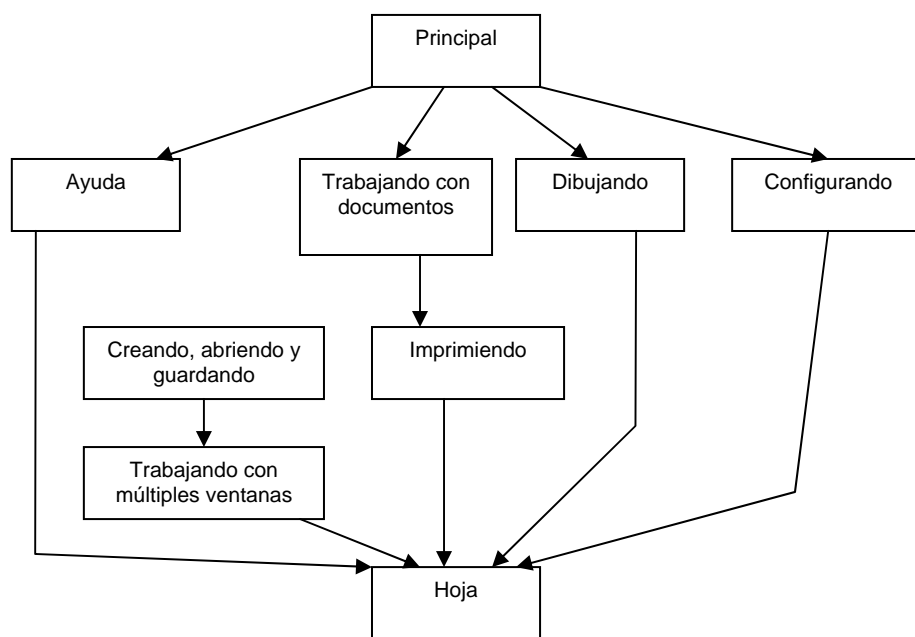


Figura 25. El concepto de enrejado

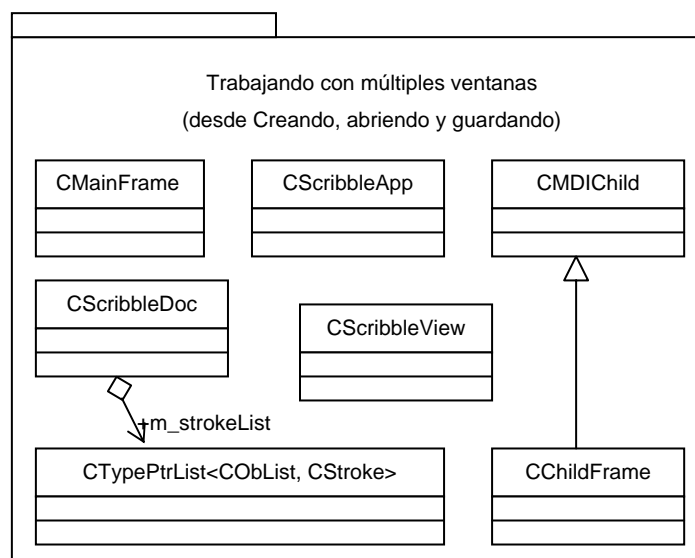


Figura 26. Trabajando con paquetes

## **5.2. Ingeniería Inversa basada en diseño de patrones**

Rudolf K. Keller, Reinhard Schauter, Sebastián Robitaille, Patrick Pagé [10] presentan una visión general de un ambiente de ingeniería inversa basado en la descripción de diseño de patrones definidos por Gamma. El propósito es introducir la ingeniería inversa basada en patrones como una técnica de valor para comprender el software y así refutar la sostenida creencia que el diseño de patrones únicamente es importante durante la ingeniería directa.

El propósito del ambiente SPOOL (características deseables que se separan en el diseño de orientado a objeto) de ingeniería inversa es ayudar a entender el software de las organizaciones a través de patrones. Esto consiste de técnicas y herramientas para la captura del código fuente, un repositorio de diseño y la funcionalidad de recuperación de patrones de diseño y representación de diseños.

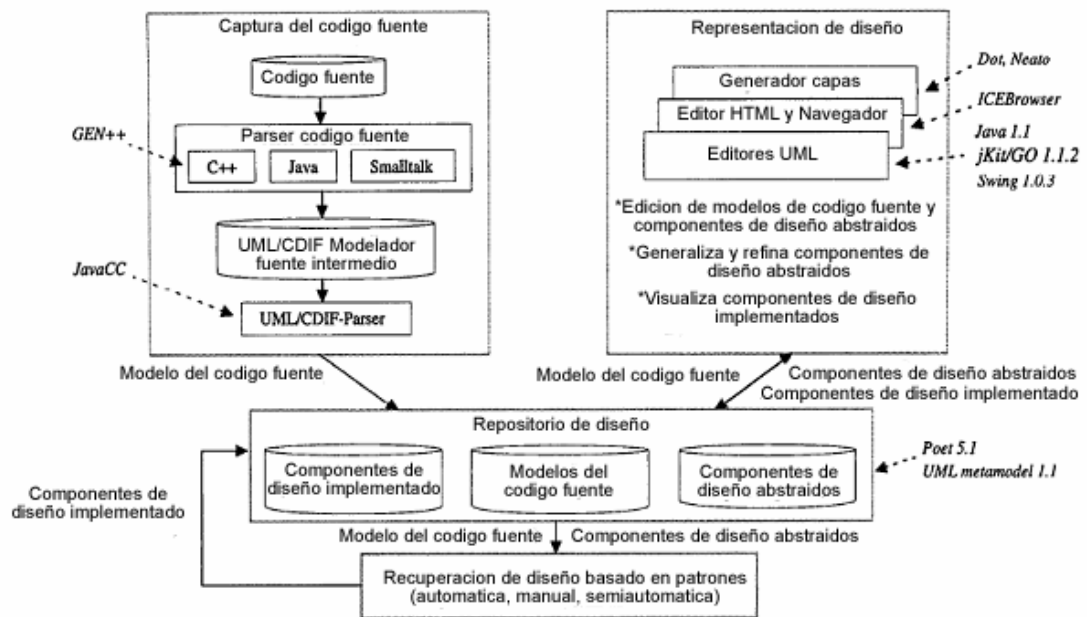


Figura 27. Visión general del ambiente SPOOL

## 1. Captura del código fuente

El propósito de la captura del código fuente es extraer un modelo inicial desde el código fuente existente. Usando GEN++, el analizador del código fuente C++, el ambiente generara representaciones basadas en código ASCII de elementos importantes de código fuente (UML/CDIF Modelador fuente intermedio), el propósito de la representación intermedia es hacer el ambiente independiente de algún lenguaje de programación específico.

## 2. Repositorio de diseño

El propósito del repositorio de diseño es proveer un almacenamiento centralizado, manipulación y consultas de los modelos del código fuente, los



componentes del diseño abstracto son recuperados y los componentes del diseño recuperados dentro de los modelos del código fuente.

El esquema del repositorio de diseño esta basado en el extendido UML metamodelo 1.1; Poet 5.1 sirve como un repositorio final para administrar base datos orientado a objetos. El esquema es representado como clases jerárquicas Java 1.1.

### 3. Recuperación de diseño basado en patrones

El propósito de la recuperación del diseño basado en patrones es ayudar a estructurar partes del diagrama de clases para armar diagramas de patrones. Son tres las técnicas:

- Recuperación de diseño Automática, relaciona la estructuración completamente automatizada de diseño de software acorde con la descripción de los patrones, los cuales son almacenados en repositorios como componentes de diseño abstracto.
- Recuperación de diseño Manual, relaciona la estructuración de diseño de software agrupando manualmente elementos de diseño, como clases, métodos, atributos o relaciones para reflejar un patrón.
- Recuperación de diseño Semiautomática combina ambas estrategias, la automática y la manual. Esto puede ser implementado como una recuperación de procesos multifases. La primera fase consiste de detección automática de bajos niveles de idiomas o

núcleo general de patrones de diseño, las fases siguientes relacionan las instancias identificadas con detalles de implementación mas específicas.

#### 4. Representación de diseño

El propósito de la recuperación de diseño es proveer la visualización interactiva y el refinamiento de los modelos de código fuente, abstracción de componentes de diseño y los componentes implementados.

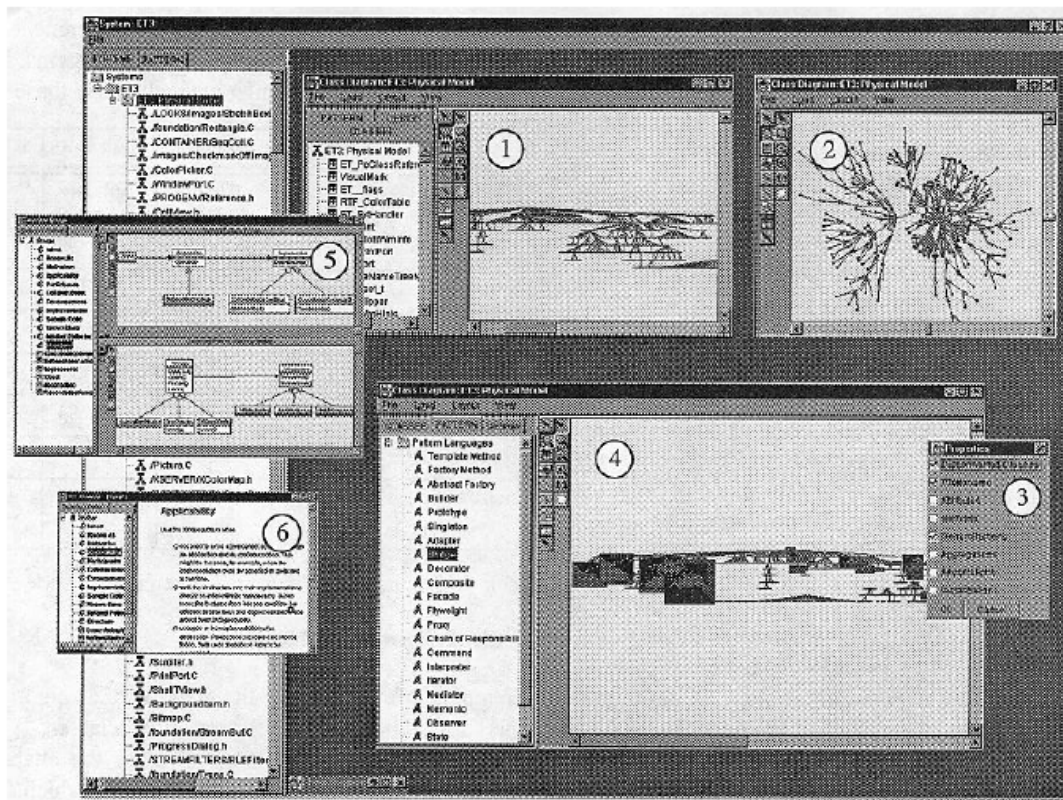


Figura 28. Interfaz grafica del ambiente SPOOL

## **CAPITULO VI**

### **6. CONCLUSIONES**

1. La ingeniería inversa es un proceso de examinación y no involucra cambiar el sistema, es considerada como un subproceso de la reingeniería, al cual si implica la modificación del sistema.
2. La ingeniería inversa puede extraer información de diseño del código fuente pero el nivel de abstracción, la completitud de la documentación, el grado con el cual trabajan al mismo tiempo las herramientas y el analista humano, y la direccionabilidad del proceso son sumamente variables.
3. La necesidad de la ingeniería inversa emerge de problemas operacionales reales en el actual sistema como por ejemplo errores debido a la mala calidad de software, al alto costo del mantenimiento debido a la carencia de documentación y a nuevos requerimientos de los usuarios.
4. Lenguajes de programación modernos están haciendo el código fuente cada vez más expresivo soportando anotaciones y reflexión, esto abre la posibilidad de recuperación de informativas vistas del código fuente.

## **CAPITULO VII**

### **7. RECOMENDACIONES**

1. El equipo de trabajo que se encargue de realizar el proceso de ingeniería inversa debería estar integrado por miembros que conozca la lógica del negocio y que tengan conocimiento de diseño de patrones.
2. La ingeniería inversa solo se debe aplicar a empresas grandes, cambiantes.

## CAPITULO VIII

### 8. REFERENCIAS BIBLIOGRAFICAS

#### 8.1. Referencia bibliográfica de un libro

[1] LIENTZ B. Y SWANSON E.B.

Software Maintenances Management

Addison Wesley, 1980

[2] PRESSMAN, Roger

5º Ed. Ingeniería del software - Un Enfoque Practico. España:

McGraw – Hill, 2002

#### 8.2. Referencia bibliográfica de un artículo

[3] BREUER P.T., y LANO K.

“Creating Specification From Code: Reverse-Engineering Techniquew”.

Journal of Sofmare Muintenance: Research and Practice, Vol. 3, pp. 145-

162, 1991

[4] PREMERLANI, W.J., y BLAHA M.R.

“An Approach for Reverse Engineering of Relational Database”. CACM,

vol. 37, Numero 5, pp. 42-49, 1994

### **8.3. Referencia bibliográfica de una pagina Web**

[5] BOOCH G, JACOBSON I Y RUMBAUGH

The UML specification documents

Santa Clara, 1997

[www.rational.com](http://www.rational.com)

[6] RUIZ Francisco y POLO Macario

Mantenimiento del Software

Universidad De Castilla-La Mancha

<http://alarcos.inf-cr.uclm.es/per/fruiz/cur/mso/trans/s1.pdf>

### **8.4. Referencia de Tesis de bachiller, licenciado, magíster o doctor**

[7] BOJIC Dragan y VELASEVIC Dusan

Reverse Engineer of Use Case Realizations in UML

Tesis Doctoral, Facultad de Ingeniería Eléctrica, Belgrado, 2001

[8] RAZO, Antonio

Reingeniería para la implementación de un Web Feature Service".

Tesis de Bachiller. Universidad de las Ameritas, Puebla, 2002

### **8.5. Referencia de Revistas Digitales**

[9] CHIKOFSKY E. y CROSS J.

Reverse Engineering and design recovery: A taxonomy.

IEEE Software, pp. 13-17, 1990

[10] KELLER R., SCHAUER R., ROBITAILLE R. y PAGE P.

Pattern-based reverse-engineering of design components

International Conference on Software Engineering

IEEE, Los Angeles, California, pp. 226-235, 1999

[11] MERLO E.

Reverse Engineering of User Interface, Proc. Working Conference on

Reverse Engineering

IEEE, Baltimore, MD, pp. 171-178, 1993

## CAPITULO IX

### 9. ANEXOS

#### ANEXO N° 01

#### CODIGO FUENTE DE LA CLASE RESERVATION DETAIL

```
<Serializable()> _  
Public Class ReservationDetail  
    Private _Agency As String  
    Private _user As String  
    Private _RGId As Integer  
    Private _ReservID As Integer  
    Private _FromDate As DateTime  
    Private _ToDate As DateTime  
    Private _Currency As String  
    Private _Status As String  
    Private _NumOfAdults As Integer  
    Private _NumOfChildren As Integer  
    Private _Note As String  
    Private _NumTran As Integer  
    Private _HotelID As Integer  
    Private _HotelName As String  
    Private _Address As String  
    Private _Category As String  
    Private _bestVal As String  
    Private _NumOfStars As Double  
    Private _RoomType As String  
    Private _Thumb As String  
    Private _DobleBed As Boolean  
    Private _FirstName As String  
    Private _MiddleName As String  
    Private _LastName As String  
    Private _HomePhone As String  
    Private _MobilPhone As String  
    Private _PriceTotal As Double  
    Private _Comi_Matr As Double  
    Private _Comi_Prin As Double  
    Private _Comi_Vend As Double  
    Private _Comi_Agen As Double  
    Private _Imp_Hotel As Double  
    Private _Imp_Agen As Double
```

```
Public Property Agencia() As String  
    Get
```



```

        Return _Agency
    End Get
    Set(ByVal Value As String)
        _Agency = Value
    End Set
End Property

Public Property User() As String
    Get
        Return _user
    End Get
    Set(ByVal Value As String)
        _user = Value
    End Set
End Property

Public Property RGId() As Integer
    Get
        Return _RGId
    End Get
    Set(ByVal Value As Integer)
        _RGId = Value
    End Set
End Property

Public Property ReservId() As Integer
    Get
        Return _ReservID
    End Get
    Set(ByVal Value As Integer)
        _ReservID = Value
    End Set
End Property

Public Property FromDate() As DateTime
    Get
        Return _FromDate
    End Get
    Set(ByVal Value As DateTime)
        _FromDate = Value
    End Set
End Property

Public Property ToDate() As DateTime
    Get
        Return _ToDate
    End Get
    Set(ByVal Value As DateTime)
        _ToDate = Value
    End Set

```

```

End Property

Public Property Currency() As String
    Get
        Return _Currency
    End Get
    Set(ByVal Value As String)
        _Currency = Value
    End Set
End Property

Public Property Status() As String
    Get
        Return _Status
    End Get
    Set(ByVal Value As String)
        _Status = Value
    End Set
End Property

Public Property NumOfAdults() As Integer
    Get
        Return _NumOfAdults
    End Get
    Set(ByVal Value As Integer)
        _NumOfAdults = Value
    End Set
End Property

Public Property NumOfChildren() As Integer
    Get
        Return _NumOfChildren
    End Get
    Set(ByVal Value As Integer)
        _NumOfChildren = Value
    End Set
End Property

Public Property Note() As String
    Get
        Return _Note
    End Get
    Set(ByVal Value As String)
        _Note = Value
    End Set
End Property

Public Property NumTran() As Integer
    Get
        Return _NumTran

```

```

        End Get
        Set(ByVal Value As Integer)
            _NumTran = Value
        End Set
    End Property

    Public Property HotelID() As Integer
        Get
            Return _HotelID
        End Get
        Set(ByVal Value As Integer)
            _HotelID = Value
        End Set
    End Property

    Public Property HotelName() As String
        Get
            Return _HotelName
        End Get
        Set(ByVal Value As String)
            _HotelName = Value
        End Set
    End Property

    Public Property Address() As String
        Get
            Return _Address
        End Get
        Set(ByVal Value As String)
            _Address = Value
        End Set
    End Property

    Public Property Category() As String
        Get
            Return _Category
        End Get
        Set(ByVal Value As String)
            _Category = Value
        End Set
    End Property

    Public Property bestVal() As String
        Get
            Return _bestVal
        End Get
        Set(ByVal Value As String)
            _bestVal = Value
        End Set
    End Property

```

```
Public Property NumOfStart() As Double
    Get
        Return _NumOfStars
    End Get
    Set(ByVal Value As Double)
        _NumOfStars = Value
    End Set
End Property
```

```
Public Property RoomType() As String
    Get
        Return _RoomType
    End Get
    Set(ByVal Value As String)
        _RoomType = Value
    End Set
End Property
```

```
Public Property Thumb() As String
    Get
        Return _Thumb
    End Get
    Set(ByVal Value As String)
        _Thumb = Value
    End Set
End Property
```

```
Public Property DobleBed() As Boolean
    Get
        Return _DobleBed
    End Get
    Set(ByVal Value As Boolean)
        _DobleBed = Value
    End Set
End Property
```

```
Public Property FirstName() As String
    Get
        Return _FirstName
    End Get
    Set(ByVal Value As String)
        _FirstName = Value
    End Set
End Property
```

```
Public Property MiddleName() As String
    Get
        Return _MiddleName
    End Get
    Set(ByVal Value As String)
        _MiddleName = Value
    End Set
End Property
```

```

        End Set
    End Property

    Public Property LastName() As String
        Get
            Return _LastName
        End Get
        Set(ByVal Value As String)
            _LastName = Value
        End Set
    End Property

    Public Property HomePhone() As String
        Get
            Return _HomePhone
        End Get
        Set(ByVal Value As String)
            _HomePhone = Value
        End Set
    End Property

    Public Property MobilPhone() As String
        Get
            Return _MobilPhone
        End Get
        Set(ByVal Value As String)
            _MobilPhone = Value
        End Set
    End Property

    Public Property PriceTotal() As Double
        Get
            Return _PriceTotal
        End Get
        Set(ByVal Value As Double)
            _PriceTotal = Value
        End Set
    End Property

    Public Property Comi_Matr() As Double
        Get
            Return _Comi_Matr
        End Get
        Set(ByVal Value As Double)
            _Comi_Matr = Value
        End Set
    End Property

    Public Property Comi_Prin() As Double
        Get

```

```

        Return _Comi_Prin
    End Get
    Set(ByVal Value As Double)
        _Comi_Prin = Value
    End Set
End Property

Public Property Comi_Vend() As Double
    Get
        Return _Comi_Vend
    End Get
    Set(ByVal Value As Double)
        _Comi_Vend = Value
    End Set
End Property

Public Property Comi_Agen() As Double
    Get
        Return _Comi_Agen
    End Get
    Set(ByVal Value As Double)
        _Comi_Agen = Value
    End Set
End Property

Public Property Imp_Hotel() As Double
    Get
        Return _Imp_Hotel
    End Get
    Set(ByVal Value As Double)
        _Imp_Hotel = Value
    End Set
End Property

Public Property Imp_Agen() As Double
    Get
        Return _Imp_Agen
    End Get
    Set(ByVal Value As Double)
        _Imp_Agen = Value
    End Set
End Property
End Class

```

## ANEXO Nº 02

### CUADRO MANTENIMIENTO DE SOFTWARE

